

CROSSTALK



Sep / Oct 2011

The Journal of Defense Software Engineering

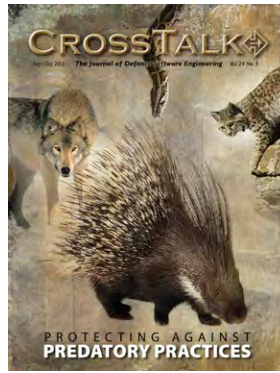
Vol. 24 No. 5

PROTECTING AGAINST
PREDATORY PRACTICES

Report Documentation Page			Form Approved OMB No. 0704-0188		
Public reporting burden for the collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington VA 22202-4302. Respondents should be aware that notwithstanding any other provision of law, no person shall be subject to a penalty for failing to comply with a collection of information if it does not display a currently valid OMB control number.					
1. REPORT DATE SEP 2011		2. REPORT TYPE		3. DATES COVERED 00-09-2011 to 00-10-2011	
4. TITLE AND SUBTITLE Crosstalk The Journal of Defense Software Engineering. Volume 24, Number 5, Sep/Oct 2011				5a. CONTRACT NUMBER	
				5b. GRANT NUMBER	
				5c. PROGRAM ELEMENT NUMBER	
6. AUTHOR(S)				5d. PROJECT NUMBER	
				5e. TASK NUMBER	
				5f. WORK UNIT NUMBER	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) 517th SMXS/MXDEA,6022 Fir Avenue,Hill AFB,UT,84056-5820				8. PERFORMING ORGANIZATION REPORT NUMBER	
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)				10. SPONSOR/MONITOR'S ACRONYM(S)	
				11. SPONSOR/MONITOR'S REPORT NUMBER(S)	
12. DISTRIBUTION/AVAILABILITY STATEMENT Approved for public release; distribution unlimited					
13. SUPPLEMENTARY NOTES					
14. ABSTRACT					
15. SUBJECT TERMS					
16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT Same as Report (SAR)	18. NUMBER OF PAGES 40	19a. NAME OF RESPONSIBLE PERSON
a. REPORT unclassified	b. ABSTRACT unclassified	c. THIS PAGE unclassified			

Departments

- 3 From the Sponsor
- 38 Upcoming Events
- 39 BackTalk



Cover Design by Kent Bingham

Protecting Against Predatory Practices

4 Cyber Espionage: Defending Against The Digital Cloak and Dagger Stealing Information Securely, Privately, and Deniably
Offensive computing is seeing a logical path of evolution where various methodologies converge into something that often functions outside the scope of the individual methods themselves, hitting us with attack vectors we barely understand, let alone for which we have a solid defense.
by Jay Bavis

6 Protecting Software Intellectual Property Against Counterfeiting and Piracy
Software counterfeiting and piracy are problems of global proportions that violate the enforcement of Intellectual Property Rights (IPR) of software developers and vendors, and thus threaten their market viability.
by Karen Mercedes Goertzel

10 Software Diversity for Future Systems Security
One security-weakening factor is related to the standardized software ecosystem that facilitates the spread of malware in systems that share common vulnerabilities.
by Abdelouahed Gherbi, Robert Charpentier, and Mario Couture

14 Creating Attack-Aware Software Applications with Real-Time Defenses
Attack-aware software applications provide attack detection and real-time defensive response with a very low false-positive rate allowing an application to detect and neutralize a threat before the attacker exploits a known or unknown vulnerability.
by Colin Watson, Michael Coates, John Melton, and Dennis Groves

19 Creating Data from Applications for Detecting Stealth Attacks
A reason for security professionals not seeming able to protect against the rapidly changing threat environment and sophisticated attacks is that they often don't have available the necessary application security data for detecting and responding to such increasingly stealthy attacks.
by C. Warren Axelrod, Ph.D.

25 Developmental Automated Testing and Software Technical Risk Assessments
Testing continues to represent the single largest cost associated with the development of sophisticated, software intensive, military systems.
by Brad Neal

29 Cyber Strategy, Analytics, and Tradeoffs
A framework of cyber tactics is suggested in terms of intended function and input and output semantics for each tactic including anticipation, detection, attribution, and counter measures.
by Don O'Neill

CROSSTALK

OUSDA(AT&L) Stephen P. Welby
NAVAIR Jeff Schwab
DHS Joe Jarzombek
309 SMXG Karl Rogers

Publisher Justin T. Hill
Advisor Kasey Thompson
Article Coordinator Lynne Wade
Managing Director Brent Baxter
Managing Editor Brandon Ellis
Associate Editor Colin Kelly
Art Director Kevin Kiernan

Phone 801-775-5555
E-mail stsc.customerservice@hill.af.mil
CrossTalk Online www.crosstalkonline.org

CROSSTALK, The Journal of Defense Software Engineering is co-sponsored by the Under Secretary of Defense for Acquisition, Technology and Logistics (OUSDA(AT&L)); U.S. Navy (USN); U.S. Air Force (USAF); and the U.S. Department of Homeland Defense (DHS). USD(AT&L) co-sponsor: Deputy Assistant Secretary of Defense for Systems Engineering. USN co-sponsor: Naval Air Systems Command. USAF co-sponsor: Ogden-ALC 309 SMXG. DHS co-sponsor: National Cyber Security Division in the National Protection and Program Directorate.

The USAF Software Technology Support Center (STSC) is the publisher of **CROSSTALK** providing both editorial oversight and technical review of the journal. **CROSSTALK's** mission is to encourage the engineering development of software to improve the reliability, sustainability, and responsiveness of our warfighting capability.

Subscriptions: Visit <www.crosstalkonline.org/subscribe> to receive an e-mail notification when each new issue is published online or to subscribe to an RSS notification feed.

Article Submissions: We welcome articles of interest to the defense software community. Articles must be approved by the **CROSSTALK** editorial board prior to publication. Please follow the Author Guidelines, available at <www.crosstalkonline.org/submission-guidelines>. **CROSSTALK** does not pay for submissions. Published articles remain the property of the authors and may be submitted to other publications. Security agency releases, clearances, and public affairs office approvals are the sole responsibility of the authors and their organizations.

Reprints: Permission to reprint or post articles must be requested from the author or the copyright holder and coordinated with **CROSSTALK**.

Trademarks and Endorsements: **CROSSTALK** is an authorized publication for members of the DoD. Contents of **CROSSTALK** are not necessarily the official views of, or endorsed by, the U.S. government, the DoD, the co-sponsors, or the STSC. All product names referenced in this issue are trademarks of their companies.

CROSSTALK Online Services:
For questions or concerns about crosstalkonline.org web content or functionality contact the **CROSSTALK** webmaster at 801-417-3000 or webmaster@luminpublishing.com.

Back Issues Available: Please phone or e-mail us to see if back issues are available free of charge.

CROSSTALK is published six times a year by the U.S. Air Force STSC in concert with Lumin Publishing <luminpublishing.com>. ISSN 2160-1577 (print); ISSN 2160-1593 (online)

CROSSTALK would like to thank
DHS for sponsoring this issue.

Protecting Our Cyber Infrastructure From Predatory Practices

Part of our role at DHS is to better enable all stakeholders to secure their part of cyberspace. Given that our adversaries will exploit even the smallest weakness, identifying and mitigating exploitable weaknesses before they become a pathway for attack is vital to the defense against predatory practices. One weak link in the chain can compromise an entire software application and degrade our enterprise capabilities.

Organizations must understand their information asset vulnerabilities. In order to assess the nature and extent of these vulnerabilities, organizations must first collect a consistent set of metrics. The Federal Government is collecting metrics with the help of the CyberScope Initiative, which mandates that federal civilian agencies report cybersecurity data using standardized formats. The CyberScope application is a web-based interactive tool that allows agencies to report data that complies with Federal Information Security Management Act (FISMA) rules. Ultimately, this tool helps federal agencies identify weaknesses, thus enabling the cyber enterprise to better defend against predatory attackers by making their assets more resilient.

Successful collection and analysis of metrics relies on standards. One standardized reporting format, a dictionary of software flaws called the Common Weakness Enumeration (CWE), provides required information feeds for FISMA reporting. The DHS National Cyber Security Division, through its Software Assurance (SwA) program, facilitates public-private collaboration that advances CWE. CWE now includes the Common Weakness Risk Analysis Framework (CWRAF) and the Common Weakness Scoring System (CWSS), which provide consistent, flexible means for identifying and mitigating the highest priority risks. CWRAF enables users to incorporate the CWSS scoring criteria to identify the most exploitable software fault patterns for web applications, control systems, embedded systems, end-point computing devices, operating systems, databases, storage systems, enterprise system applications, and cloud computing services. Identifying and mitigating exploitable software weaknesses before they become vectors of attack are some of the most effective means for addressing predatory practices.

The SwA program also sponsors other security automation enumerations and languages that enable consistent, interoperable reporting among IT security tools and services. These open, voluntary standards accelerate actionable information exchange within the incident response community. The set of standards include the Open Vulnerability and Assessment Language, the Common Attack Pattern Enumeration and Classification, and the Malware Attribute Enumeration and Characterization. These languages and enumerations are resources that help software engineers and enterprise IT security managers to identify and correct errors early in and throughout the software lifecycle. Integrating security into the software lifecycle is a critical practice and vital to the safety and resilience of our software-enabled systems.

Powerful tools and services that use these enumerations and languages already exist—and the ranks of users are growing quickly. This CrossTalk issue will alert participants in the software lifecycle of certain predatory practices and enable them to arm themselves with means that efficiently find and assist in mitigating software security flaws.

Roberta “Bobbie” Stempfley

*Acting Assistant Secretary,
Office of Cybersecurity and Communications
Department of Homeland Security*



Cyber Espionage

Defending Against The Digital Cloak and Dagger Stealing Information Securely, Privately, and Deniably

Jay Bavisi, EC-Council

Abstract. As multi-disciplined approaches to offensive computing take shape, combining everything from cryptographic protocols to anonymous communication channels to malicious software, we are starting down a logical path of evolution where various methodologies converge into something that often functions outside the scope of the individual methods themselves, hitting us with attack vectors we barely understand, let alone for which we have a solid defense.

In the case of cyber espionage, covert offense is the underlying goal of an adversary. The subtlety of the attack is paramount since drawing too much attention may alert the victim and result in immediate countermeasures, thus revealing, and halting, any attack efforts.

To ensure the secrecy and privacy of these activities, an adversary might deploy malicious software that is capable of stealing information without revealing what it is that the adversary is looking for or what the adversary has taken—which is known in cryptography as Private Information Retrieval (PIR).

First, let us look at the basic algorithmic model of a PIR-based attack, based on the foundation that an adversary is attempting to privately retrieve information from a database without the database administrator's knowledge. Suppose we have a database consisting of n entries where each entry holds a single binary bit; the database can be represented by a bit string, $B = b_1 b_2 \dots b_n$.

The adversary needs to submit a query (i) to the database administrator, where $1 \leq i \leq n$, such that i is not revealed to the database administrator, while b_i is returned as a response to that query. In this manner, b_i has been privately retrieved from the database by the adversary.

Consider the following simplified attack model within the context of corporate espionage: An adversary—let us say an actual employee, an insider—is disgruntled by the possibility that another employee is being overly compensated. The adversary designs a piece of malicious software, in the form of a virus that contains a tag string. Depending on how the database is indexed, this tag string could be programmed to look for the salary information of a given employee, for example.

With that in mind, when the virus matches its tag string to an entry in the database, it knows it has found its target and internally stores the data within itself. Because this data is stored clearly and could potentially be discovered by anyone who obtains the virus, the adversary would encrypt the data using a public key, for which he possesses a corresponding private key

that will be used to decrypt the data when the adversary obtains the virus. This is referred to as a cryptovirus, which can even make use of the cryptographic API (e.g., Microsoft's CryptoAPI) within the victim's host system, thus allowing the cryptovirus to be even more lightweight, by eradicating the need for an onboard cryptosystem.

In addition, because snapshots of a virus may reveal the tag string, which could also be obtained by anyone, the aforementioned PIR algorithm is used to prevent this from occurring by concealing the tag string after the adversary retrieves the data of interest. Of course, we can imagine applications outside of corporate espionage—governments, militaries, financial institutions, etc.—and this is just a method for stealing specific, isolated data. What if we could privately steal passwords and covertly obtain access to hordes of information? We can.

Take the notion of a cryptovirus and mesh it with the notion of a Trojan horse; now you have cryptotrojan. To further the idea of stealing information, it would be even more ideal if the adversary had some sense of deniability, such that even if his activity is noticed, it cannot be proven beyond a shadow of doubt that he is actually attacking—despite the use of auditing or logging mechanisms—thanks to the combination of cryptography and mix network channels that provide anonymity.

Not only that, but the cryptotrojan can be designed in such a way as to prevent the victim, upon possible discovery of the cryptotrojan, from determining which passwords, if any, have been stolen or even whether or not the cryptotrojan has stolen anything at all. The implications of this are substantial, as we have now combined malicious software, communication channels, and cryptographic algorithms to build a platform for deniable espionage—cyber spies that can not be caught, even if they are caught.

We are already seeing, in the real world, evidence of the adversaries' consciousness of malicious software that uses cryptography to cover itself with advanced worms such as Conficker, implementing state-of-the-art cryptographic algorithms to prevent the hijacking of payloads. And, with other recent large-scale attacks, as we have seen with the Stuxnet worm and the espionage network, GhostNet, it is not hard to hypothesize the significant impact it would have should the malware of the future employ these covert techniques to evasively wreak havoc, and leave us scratching our heads as to what just happened, and who we are to thank for it.

Detecting Abnormal and Cryptographic Code

Working from the common knowledge that a virus often appends itself to an object such as an executable, and changes the point of entry to itself, you now have a heuristic by which to detect it. Certain detection techniques operate on the premise that custom malicious software will produce statistical discrepancies when they manifest themselves through otherwise ordinary binary executables; these techniques seek out such anomalies and facilitate the detection of potentially malicious code.

Heuristic analysis can be used to pinpoint probable malicious code the same way it can also be used to pinpoint the existence of cryptographic code. Previously, we discussed the use of pub-

lic and private cryptographic keys to conceal the data pilfered by the cryptovirus. Because common asymmetric cryptosystems are based on the integer factorization problem, which is the issue of determining the prime factorization of a given integer, cryptographic APIs will often test primality by performing trial division on small primes and store a list of the results which could potentially be discovered using string matching.

Taking this a step further, because asymmetric cryptography's arithmetic-intensive nature can render many implementations inefficient, an algorithm known as the Karatsuba algorithm can be used to speed up the multiplication between two large numbers. Along with detecting the presence of such efficiency-boosting algorithms for key generation, the actual presence of public keys themselves can be investigated in long bit strings via specific algebraic methods, or in large programs, via more general statistical methods. Because cryptographic keys depend on randomness, they will exhibit much more entropy than surrounding data, which is more structured and redundant. In a visual sense, cryptographic keys would appear to be noisier than surrounding data.

Unfortunately, this brings to light the double-edged nature of cryptography, which is typically used as a defense mechanism for preserving goals like confidentiality and integrity. On top of that, cryptography is arguably the strongest link in any

system. This strength translates directly into the malicious use of cryptography, making intrusion detection and response particularly difficult.

Spies of old, despite their tools and savvy, would, if caught, fear for their own life—be it spent in prison or swiftly ended by other means. Denial does not work too well. For digital spies however, it can, and thanks to the nefarious use of our own weapons against us, there is not always a solution. This frontier is relatively new and sparsely explored, and judging by the caliber of what is possible now, it is likely that we will be even more impressed, or depressed, by what lies ahead, depending on which side you are on. ♦

ABOUT THE AUTHOR



Jay Bavisi is the co-founder and president of EC-Council, the governing body of the world-renowned Certified Ethical Hacker CEH program. A distinguished author and speaker on information security, he has given keynote speeches at international conferences, been invited to lecture at international corporations and academic institutions, contributed numerous articles to major technical publications, and had his views sought after by internationally acclaimed media giants, such as The Wall Street Journal, CNN, TIME Magazine, and USA Today. Mr. Bavisi is a law graduate from the University of Wales, College of Cardiff, having an LLB (Hons), Barrister-at-Law from Middle Temple, London.

WANTED

Electrical Engineers and Computer Scientists *Be on the Cutting Edge of Software Development*

The Software Maintenance Group at Hill Air Force Base is recruiting **civilian positions** (U.S. Citizenship Required). Benefits include paid vacation, health care plans, matching retirement fund, tuition assistance and time off for fitness activities. **Become part of the best and brightest!**

Hill Air Force Base is located close to the Wasatch and Uinta mountains with many recreational opportunities available.



Send resumes to:
phil.coumans@hill.af.mil
or call (801) 586-5325

Visit us at:
<http://www.309SMXG.hill.af.mil>

Protecting Software Intellectual Property Against Counterfeiting and Piracy

Karen Mercedes Goertzel, Booz Allen Hamilton

Abstract. Software counterfeiting and piracy are problems of global proportions that violate the enforcement of Intellectual Property Rights (IPR) of software developers and vendors, and thus threaten their market viability. Moreover, software counterfeiting provides violators with the opportunity to modify and augment the duplicated software code in undesirable ways, including insertion of malicious logic, backdoors, and exploitable vulnerabilities. Technological solutions to these challenges focus on making software authenticity easier to verify, making software more difficult to counterfeit, and making software distribution processes harder to subvert. But the software industry and governments worldwide recognize that technology is not the sole answer to reducing piracy and counterfeiting. They are focusing their efforts both on technological research but even more on IPR legislation, trade agreements, enforcement, “best practices” and awareness that both complement and reinforce technological approaches.

In DoD, DHS, the intelligence community, and other security-focused organizations, a frequently discussed supply chain threat to software is the rogue developer or distributor who embeds malicious logic, backdoors, or intentional vulnerabilities in source code or binary executables prior to releasing them to customers. (From here on “malicious code” should be understood to refer collectively to all three types of inclusions.)

The threat of counterfeiting is another concern, especially counterfeit integrated circuits and network devices, and increasingly software. Three factors drive this concern: (1) counterfeits are often found to be less dependable than the original products they are intended to copy; (2) counterfeiting occurs outside the legitimate supply chain, thus obscuring supply chain transparency and traceability of product, product pedigree, and product provenance; and (3) counterfeiting violates the IPR of the copied product’s legitimate vendor/developer and threatens their business viability.

For software, the terms “counterfeiting” and “piracy” are often used interchangeably, so the distinction between them has become obscure. For purposes of this article, piracy means the act of illicitly distributing copies of software without a legitimate license. Counterfeiting means the substitution at any point in the supply chain of legitimate license-bearing software with an illicit replica. Counterfeit replicas are often modified or augmented, e.g., through removal of anti-tamper protections or insertion of malicious code, thus tying the first concern about malicious code with the second about illicit copies. Pirated software is sometimes, but not always, counterfeit. Sometimes it is directly copied without alteration.

Given the three factors cited above, the desire to know a software product’s true pedigree (origin) and provenance (tools/processes used to develop and move it through its supply chain) exceed the imperatives of IPR enforcement, also enabling the purchaser to assess the trustworthiness of the parties known to be involved in creating and distributing all components of the software product. From this knowledge can be inferred some level of confidence in the software’s dependability, trustworthiness, and authenticity. Pedigree/provenance indicators also provide an “audit trail” by which accountability can be traced back to those parties.

Currently, software pedigree analysis relies on “fingerprints” or “signatures” of acquired source code, which are compared against those of known, legitimate open source or (where applicable) closed source code. A pedigree analysis tool determines whether a given instance of source code has been copied from a known code base and is compliant with the code base’s licensing restrictions. The tools also reveal whether the copied code was modified (e.g., through malicious insertions). Palamida, Protecode, and Blackduck Software offer commercial toolsets and/or services for source code pedigree analysis.

Theoretically pedigree analysis should be possible for source code derived through reverse engineering of binary executables. In practice, however, reverse engineering produces reconstituted source code that deviates significantly from the original code-as-written. Any “fingerprint” of the reconstituted source code will be useless as a basis for comparison with the original code. Moreover, use of any pedigree analysis approach that requires reverse engineering would require a vendor willing to: (1) provide access to their source code (as a basis of comparison); and (2) permit the violation of their software license restrictions that prohibit reverse engineering of their executables. Neither of these is likely.

In *Software Piracy Exposed*, the authors state that, “Insiders are constant suppliers to the piracy market” [1]. Digital piracy (warez) groups constantly seek cooperative software industry (as well as music, games, and movie industry) insiders to act as “suppliers” who will copy and upload their employers’ software products to warez sites for other group members to download. Moreover, every warez group has at least one industry insider actively working for it. Often, these insiders learn about new releases of products well before those products are officially released, and provide illicit pre-release copies to the warez sites. Pirated pre-release versions of software are often detected by their vendors because they include CD keys or serial numbers that get stripped out of official release distributions.

Another source of pirated software is the low-wage worker in a media packaging warehouse, easily induced by promises of payment to steal and provide CDs to warez groups; a premium is paid for master replication discs.

Software counterfeiting extends to manuals, brochures, labels, certificates of authenticity, and license agreements that come with software products. Direct copies of software are considered counterfeit if any of these ancillary materials is counterfeit. Counterfeiting of ancillary materials may involve copying and slightly altering legitimate product data or substituting persuasive false data that obscures a product’s pedigree or hides a negative aspect of the original or counterfeit version. For example, a product that has no vendor or third-party certification (e.g., Open Group

certifications, Novell YES certification, etc.) may be distributed by a counterfeiter with one or more forged certificates.

The supplier of a counterfeited/pirated product is under no obligation to support customers who wittingly or unwittingly buy a counterfeited/pirated copy. In some cases, however, legitimate suppliers may be duped into supporting an illicit copy if it includes a valid serial number that is not yet registered. In such cases, the illicit copy's owner can register that serial number and obtain support.

Other concerns raised by counterfeiting/piracy include the potential loss of a legitimate vendor's reputation and brand integrity. The convoluted and complexity of the Information and Communications Technologies (ICT) supply chain is such that even original manufacturers may receive counterfeit parts/components or pirated software copies from their suppliers, and may integrate them into their larger products, so the result is a product that is a hybrid of genuine and counterfeit content. If such products are discovered by customers to be defective, or to include malicious insertions, the original manufacturer will be at fault, regardless of its lack of direct responsibility for the counterfeiting or piracy. Manufacturers may need to be even more vigilant as acquirers of ICT parts/components than purchasers of their end products are.

Counterfeits are also frequently offered by independent distributors, brokers, and other "gray markets," which represent "channel diversion" from the legitimate market. This is a particular risk when customers consider lowest price their first or only criterion for source selection. Whenever such customers purchase illegitimate products from gray market sources, the market share of the original manufacturers of legitimate products, and their authorized distributors, is eroded. The financial health of legitimate suppliers is a concern, because product obsolescence (including when manufacturers go out of business) is a major driver for the counterfeiting industry. Indeed, gray market outlets are often the only available sources for obsolete parts.

Gray markets are usually unintentional violators, such as unauthorized distributors/retailers, online auction sites, open source and software reuse repositories, third-party download sites, thrift shops, garage sales, dollar stores, and other low-price retailers. By contrast, black markets are intentional IPR violators. Example black market outlets are software piracy/warez download and peer-to-peer file sharing sites, as well as hackers that use cross-site scripting to surreptitiously redirect customer browsers away from legitimate software download sites to illegitimate, usually malicious, replica sites.

Protecting Against Counterfeiting and Piracy

The software vulnerability most often blamed for making counterfeiting and piracy possible is lack of robust software copy protections. Also decried is a lack of anti-tamper protections to prevent modification or augmentation of copied code. Tamper proofing and tamper deterrence can be expensive to implement, so products mainly aimed at the consumer market are generally limited to having anti-tamper mechanisms applied to their packages, but not to the software itself. By contrast, software intended for the business market is more likely to include direct tamper deterrence and evidence mechanisms.

Whether distributed on physical media or via a digital trans-

mission or download, software, its packaging, its storage facility (warehouse or download server), and its distribution channel (physical transportation mechanisms or network-based download/transmission mechanisms) need to be secured from end to end.

Trusted distribution for software should:

- * Enable customers to authenticate the supply source. For digital downloads, the customer should be able to detect whether a cross-site scripting attack has surreptitiously rerouted the browser to a counterfeit download server.
- * Tamper proof the software (via hash, digital code signature, and/or digital watermark).
- * Include tamper deterrence/evidence mechanisms in physical packaging.
- * Use read-only media for shipping software and documentation.
- * Authenticate the acquirer, ideally before a software download (e.g., through use of supplier-provided download key), but definitely at installation time and/or first execution time (through entry of supplier-provided installation and execution keys). Increasingly, software executables are encrypted before download or copying to distribution media, so only licensed customers who have the vendor-provided cryptokey can decrypt the software.
- * Separate distribution channels for the software and any keys needed to decrypt, install, or execute it. For example, the supplier may require an online buyer to provide an e-mail address to which such keys will be sent. Authenticated distribution channels and separation of paths can be accomplished for physical distributions using registered mail (or a shipping service) and requiring the recipient to sign for the package containing the software. Associated keys should be sent in a separate physical shipment (ideally by a different shipping service).
- * Make sure servers and network connections/sessions involved are similarly protected using cryptographic and access control means because online software sales and download transactions are susceptible to the same vulnerabilities and threats as all other networked computer-based online purchasing transactions.

In 1994, the Bellcore Trusted Software Integrity system [2] established the use of a combined cryptography hash function with an X.509 digital signature affixed to a software executable to protect the integrity of that software during its distribution over the Internet. Today, much the same approach is used to protect integrity of software distributions, but the additional concern over software piracy means that digital signatures and hashes are increasingly augmented with software IP protections such as code obfuscation and software watermarking.

Digital asset management systems are used to track, with accountability, the movement of and access to software assets. Such systems employ digital watermarks and other techniques to capture the time and location of every access. Digital software watermarking involves steganographic hiding of steganographic messages (signals) in the software code. Watermarks enable the purchaser to verify the authenticity of the software's supplier and to detect (though not deter) tampering [3]. If a watermark is unique to a particular copy of the code, it also acts as a fingerprint which, if detected on another copy of the software, indicates that one of the two copies was pirated.

Researchers are investigating strong watermarking techniques to degrade performance or prevent execution of illicit software copies. Software watermarking technology thus also provides a means of license enforcement.

In the movie industry, strong watermarks are applied to film content that will be digitally distributed to prevent illicit copying. If a strong watermark is removed or tampered with before the illicit copy is made, the film content will be altered enough to significantly degrade viewing quality. Researchers are investigating strong watermarking techniques to degrade performance or prevent execution of illicit software copies. Software watermarking technology thus also provides a means of license enforcement.

To deter reverse engineering of software as the basis for counterfeiting-related augmentation/modification, code obfuscation is often used. Code obfuscations can be applied to source code, bytecode, object code, or binary executable code; they are alterations that obscure the purpose of the code without changing its operation. The goal is to make it difficult for a would-be reverse engineer to understand or tamper with the code in a meaningful way, or to locate and circumvent any copy-protection mechanisms in the code. A related technology, media obfuscation, physically alters the CD or DVD on which software is distributed, usually by adding a hard-to-reproduce cryptographic “taint” to the disc as it is pressed. This taint must be detected and decrypted for the software to be installable and executable. An example of an obfuscation system for software media is MLS LaserLock.

Digital code signatures applied to software executables do double duty as product authenticators and tamper indicators. Code signatures do nothing to warrant the absence of tampering or malicious code insertions that occur before the code signature is applied.

Digital Rights Management (DRM) controls are increasingly applied to software documentation, especially when delivered in PDF format. DRM embeds or overlays cryptographic access controls on protected digital content, most often to prevent cut-and-paste copying and/or printing by anyone who does not possess the necessary decryption keys. DRM is problematic when used to protect software code [4] because its white box cryptography model renders the digital certificates containing the DRM decryption keys vulnerable after they are issued. Anyone who obtains a copy of the certificate can use it to decrypt the software. This is a particular problem because DRM-protected software is often distributed without any other form of protection such as code signatures. Pirates and counterfeiters often copy the contents of DRM digital signatures and include the extracted decryption keys with the copied software; these keys can be forwarded on with each copy made. DRM products also require the protected software's installer or user to enter and verify long alphanumeric installation keys or passphrases, making it impossible to fully automate software installation.

Tethering (a.k.a. “product activation”) is used by some major software vendors (e.g., Microsoft, Adobe) to enforce software licensing and prevent piracy. Tethering links the software's license with the specific computer hardware on which it is first installed. The software records the unique identifier of the computer's CPU or Ethernet card's MAC address. The hardware identifier is then hashed and sent to the vendor's product activation server, which uses a non-invertible hash function to gener-

ate a second unique identifier that it returns to the software. The software can only execute if both hash identifiers are present. Copying the installed software to another CPU will necessarily “break” the hardware-software binding and cause execution to fail. Tethering evolved from the “dongle” concept. A dongle was a device required by some software vendors in the 1990s and early 2000s; it had to be connected to a bus on the computer before the installed software could execute. If the dongle could not be detected by the software's boot routine, execution would fail. Dongles were abandoned by most software vendors due to inconvenience and unreliability.

Some vendors now ship software with a hardware or software guard module that monitors the running software to ensure its authenticity and integrity. Software guards can be embedded in the software code itself, and can perform simple tasks such as checksum calculation/validation and code repair. Recently, vendors have been implementing groups of cooperating guards that work in tandem to perform more sophisticated security tasks. For example, a software product may have different parts of its code protected by guards that apply different checksums to different parts of the software. If an attacker manages to crack one checksum, the others will remain intact unless they are all cracked as well. The time and effort needed to crack so many checksums may be enough to act as a deterrent to most hackers.

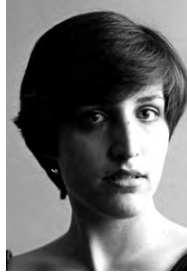
Some software products include monitoring programs that perform execution-time checks to ensure the software has been legitimately licensed. The monitor reports any license violations to the software vendor or a third party. Such monitoring programs often perform additional spyware functions, such as tracking and reporting user activities to advertisers who have paid the software vendor for use of the vendor's customers' data.

Note that there are inherent weaknesses in cryptographic solutions such as digital signatures and certificates for code signing. Non-white box encryption of code and many other protections depend on the presence of an active network channel between the system on which the software is installed and a server operated by the software vendor (e.g., for certificate validation, host attribute checking, etc.). Such a network dependency poses non-trivial difficulties for software on platforms that operate offline. Add to this the inherent complexity of cryptokey distribution and key and certificate revocation. And finally, all software protection mechanisms cannot be fully trusted because, as with all ICT, there is no way to verify that the tools that implemented/applied the software protection mechanisms were authentic and trustworthy.

IPR Enforcement Efforts

In 2005, KPMG and the Alliance for Gray Market and Counterfeit Abatement published a study that found 10% of information technology products likely to be counterfeits, representing a conservatively estimated loss in revenues and outlays due to returns and exchanges of \$10 billion annually [5]. Given the size of potential losses involved, governments and industry trade associations are actively pursuing technology, legislative, regulatory, and trade-related mechanisms for protecting IPR, catching and prosecuting IPR violators, and reducing incidence of counterfeiting and piracy overall. A small but significant portion of this activity focuses on software.

ABOUT THE AUTHOR



Karen Mercedes Goertzel, CISSP, leads Booz Allen Hamilton's Security Research Service. An expert in the insider threat to information systems, ICT supply chain risk management, software assurance, and cross-domain information sharing, she has performed in-depth research and analysis for customers in DoD, intelligence community, civil agencies, and industry in the U.S., the UK, NATO, Australia, and Canada. She was lead author/editor of the DoD Information Assurance Technology Analysis Center's 2010 State-of-the-Art-Report entitled *Security Risk Management for the Off the Shelf Information and Communications Technology Supply Chain*.

Karen Mercedes Goertzel
Booz Allen Hamilton
 c/o P.O. Box 694
 Merrifield, VA 22116-0694
 Phone: 703-698-7454
 E-mail: goertzel_karen@bah.com

REFERENCES

1. Craig, Paul and Ron Honick. *Software Piracy Exposed*. Burlington, MA: Syngress Publishing, 2005.
2. Rubin, Aviel D. "Trusted Distribution of Software over the Internet", in *Proceedings of the 1995 Internet Society Symposium on Network and Distributed System Security* (San Diego, CA: February 1995), pages 47-53. <<http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.45.4731&rep=rep1&type=ps>> (The Betsi technology was also submitted by Rubin as Internet Engineering Task Force Request For Comment 1805, "Location-Independent Data/Software Integrity Protocol", June 1995. <<http://tools.ietf.org/html/rfc1805>>.)
3. Myles, Ginger. "Using Software Watermarking to Discourage Piracy". *ACM Crossroads*, 10:3 (Spring 2004) <<http://www.acm.org/crossroads/xrds10-3/watermarking.html>>.
4. Djekic, Petar and Claudia Loebbecke. "Software Piracy Prevention through Digital Rights Management Systems". *Proceedings of the Seventh IEEE International Conference on E-Commerce Technology*. Munich, Germany, July 2005.
5. KPMG and the Alliance for Gray Market and Counterfeit Abatement. "Managing the Risks of Counterfeiting in the Information Technology Industry" (2005) <http://www.agmaglobal.org/press_events/press_docs/Counterfeit_WhitePaper_Final.pdf>.
6. Lau, Justine. "Software Groups Hail China Piracy Verdict". *The Financial Times* (21 August 2009)
7. Einhorn, Bruce. "Microsoft Has Hope in Asian Piracy Fight". *Business Week* (27 February 2009) <http://www.businessweek.com/globalbiz/content/feb2009/gb20090227_551561.htm>.

ADDITIONAL READING

- * Atallah, Mikhail J., Eric D. Bryant, and Martin R. Stytz. "A Survey of Anti-Tamper Technologies". *CrossTalk: The Journal of Defense Software Engineering* (November 2004) <<http://www.crosstalkonline.org/storage/issue-archives/2004/200411/200411-Atallah.pdf>>.
- * Castro, Daniel, Richard Bennett, and Scott Andes. "Steal These Policies: Strategies for Reducing Digital Piracy". (December 2009) <<http://www.itif.org/index.php?id=324>>.
- * Collberg, Christian and Jasvir Nagra. *Surreptitious Software: Obfuscation, Watermarking, and Tamperproofing for Software Protection*. Hoboken, NJ: Wiley, 2009.
- * Goertzel, Karen Mercedes, Theodore Winograd, et al., for Defense Technical Information Center Information Assurance Technology Analysis Center. *Security Risk Management for the Off-the-Shelf Information and Communications Technology Supply Chain* (Unclassified//Distribution limited to U.S. government and government contractors). Herndon, VA: 2010.
- * National Computer Security Center. *Guide to Understanding Trusted Distribution in Trusted Systems* (NCSC-TG-008, December 1988) <<http://www.fas.org/irp/nsa/rainbow/tg008.htm>>.
- * Software Piracy and Measures for Prevention of Piracy. <<http://legalsutra.org/665/software-piracy-and-measures-for-prevention>>.
- * Yacobi, Yacov, Microsoft Research, and Gideon Yaniv, COM Academic Studies. "Counterfeiting and anti-counterfeiting of software and content". *Proceedings of the 8th ACM Workshop on Digital Rights Management*, Alexandria, VA (October 2008): 53-58.

Software counterfeiters and pirates are being caught and prosecuted under increasingly strict IP protection laws thanks to national and international law enforcement efforts such as the multinational Operation Fastlink and U.S. Operation Higher Education. The latter resulted in the confiscation of hundreds of illegal software online distribution hubs; the removal of more than \$50 million worth of illegally copied software, games, movies, and music from illicit distribution channels; and the conviction of more than 60 perpetrators in the U.S. alone.

While many prosecuted counterfeiters and pirates are small-scale operations, some are impressively large and organized. For example, from 2005 to 2007 a joint FBI/China Ministry of Public Security law enforcement operation named "Summer Solstice" unearthed an organized crime ring in the U.S. and China responsible for hundreds of thousands of counterfeits of Microsoft, Symantec, and other vendors' software. Summer Solstice operations seized more than 337,000 counterfeit software CDs and certificates of authenticity (estimated retail value \$502 million) plus eight high-quality master replication discs and dismantled retail facilities and factories of more than 14 major counterfeiting organizations. Also heartening is the fact that under global pressure to increase enforcement of international IPR laws and agreements, China has finally become more diligent in pursuing software counterfeiters. In 2009, for example, a district court in Suzhou heavily fined and imprisoned four men found guilty of distributing counterfeit versions of Windows XP and other software programs over the Internet [6]. Similar crack-downs have taken place in other countries where counterfeiting and piracy are notorious (e.g., Thailand) [7].

A number of organizations devoted to IPR issues and initiatives have emerged in recent years, including the Federation Against Software Theft and Japan's Association for Copyright of Computer Software. Virtually every trade association representing the software industry (e.g., Business Software Alliance, Software and Information Industry Association, Entertainment Software Association, Content Delivery and Storage Association, Entertainment and Leisure Software Publishers Association) has instituted IPR enforcement, best practice, certification, and awareness efforts. Moreover, broader IPR initiatives, ranging from legislative initiatives (e.g., Piracy and Counterfeiting Amendments Act of 1982, Digital Millennium Copyright Act, Trade Enforcement Act of 2009, Pro-IP Act of 2007, Section 107 of the UK Copyright Designs and Patents Act of 1988) to government awareness and enforcement initiatives (e.g., 1998 Presidential Executive Order 13103 on "Computer Software Piracy," Organization for Economic Cooperation and Development Project on Counterfeiting and Piracy, National IPR Coordination Council, National Intellectual Property Law Enforcement Coordination Council, FBI anti-piracy warnings, NIST National Software Reference Library) to international trade initiatives (e.g., World Trade Organization Agreement on Trade-Related Aspects of IPR, Anti-Counterfeiting Trade Agreement, World Intellectual Property Organization Copyright Treaty of 1996, Asia-Pacific Economic Cooperation Anti-Counterfeiting and Piracy Initiative) all emphasize ICT-related IPR enforcement (hardware and software) as key to continuing the health not just of national economies, but of the global economy. ♦

Software Diversity for Future Systems Security

Abdelouahed Gherbi, École de technologie supérieure
Robert Charpentier, Defence Research and Development Canada
Mario Couture, Defence Research and Development Canada

Abstract. Software security represents a major concern as cyber attacks continue to grow in number and sophistication. One security-weakening factor is related to the standardized software ecosystem that facilitates the spread of malware in systems that share common vulnerabilities. In this overview article, the main concepts associated with diversity and software redundancy are described in the perspective of improving attack resistance. The remarkable progress made in this area, where commercial implementations are now emerging, is also highlighted.

1. Introduction

The security of information systems remains an extremely critical issue despite the good progress that was made in the last 10 years in the fields of software quality and system reliability. It seems that defensive measures cannot catch up to the continuous growth of cyber threats that are not only increasing in number but also in sophistication and scale [1, 2].

It remains extremely difficult to produce fault-free software despite the rigorous quality controls that are generally part of the software development process. These residual faults constitute dormant vulnerabilities that would eventually end up being discovered by malicious attackers and exploited to carry out cyber attacks. Moreover, in order to ease the system management, reduce the configuration errors, and achieve portability, most of the systems used nowadays run substantially similar software. This is called information technology monoculture [3, 4]. As a consequence, these systems share similar vulnerabilities that facilitate malware propagation and enable large-scale exploitation of these common vulnerabilities.

The Canadian Forces like most armed forces around the world, are very much concerned by the “conjugation of these risks factors”: the increased threat capabilities aimed at vulnerable infrastructures combined with society’s dependency on information sharing. Recognizing that cyber attacks are inevitable in the future, a shift from the traditional defensive strategies toward more proactive measures can now be observed. This includes: (a) more rigorous monitoring for earlier attack detection; (b) the capture of legal evidence (cyber forensics) to enable post-event investigation; (c) some semi-automated responses to the most likely attacks; and (d) pre-programmed recovery strategies to minimize the impact of successful attacks.

Among the technologies that have the potential of mitigating the cyber attack risks, “software redundancy” that includes “component diversity” appears to be one of the rare technologies promising an order-of-magnitude increase in system security. The

basic idea is simply to have critical systems implemented in two (or more) instances using sufficiently different sub-systems (e.g., Linux and Unix BSD) so the same dormant vulnerability does not exist in both redundant systems, making it impossible for the attackers to exploit the same vulnerability in both instances simultaneously. Not only does such architecture offer attack resistance, it also greatly improves the monitoring of transactions and the early detection of abnormal behavior by the comparison of both executions. It also enables continuity of services since the replica can handle the user’s requests while the first system is targeted, investigated, or recovering from a recent attack.

In 2008, Defence R&D Canada initiated a study to evaluate the state-of-the-art in software redundancy implementing technological diversity to mitigate the risk associated with the IT monoculture. The amount of high-quality work that is going on in the scientific community is impressive. This short article gives an overview of the state-of-the-art in system redundancy using different types of diversity paradigms.

2. Redundancy and Diversity Combined in a Defense Mechanism

Redundancy is traditionally used to achieve fault tolerance and higher system reliability. This has proven to be valid mainly for hardware because of the failure independence assumption as hardware failures are typically due to random faults. Therefore, the replication of components provides added assurance. When it comes to software, however, failures are due to design and/or implementation faults. As a result, such faults are embedded within the software and their manifestation is systematic. Therefore, redundancy alone is not effective against software faults.

Faults embedded in software represent potential vulnerabilities, which can be exploited by external interactive malicious fault (i.e., attacks) [5]. These attacks can ultimately enable the violation of the system security property (i.e., security failure) [5]. Therefore the diversity principle can potentially be used for security purposes. First, diversity can be used to decrease the common vulnerabilities. This is achieved by building a software system out of a set of diverse but functionally equivalent components. This in turns makes it very difficult for a malicious opponent to be able to break into a system with the very same attack. Second, the ability to build a system out of redundant and diverse components provides an opportunity to monitor the system by comparing the dynamic behavior of the diverse components when presented with the same input. This endows the system with efficient intrusion detection capability.

Therefore, diversity has naturally caught the attention of the software security research community. The seminal work presented by Forrest et al. [6] promotes the general philosophy of system security using diversity. The authors argue that uniformity represents a potential weakness because any flaw or vulnerability in an application is replicated on many machines. The security and the robustness of a system can be enhanced through the deliberate introduction of diversity. Deswarte et al. review [7] the different levels of diversity of software and hardware systems and distinguish different dimensions and different degrees of diversity [8]. Bain et al. [9] presented a study to understand the effects of diversity on the survivability of systems faced with a

set of widespread computer attacks including the Morris worm, Melissa virus, and LoveLetter worm. Ammann et al. [10] report on a discussion held by a panel of renowned researchers about the use of diversity as a strategy for computer security and the main open issues requiring further research. It emerges from this discussion that there is a lack of quantitative information on the cost associated with diversity-based solutions and a lack of knowledge about the extent of protection provided by diversity.

Three main levels of security enhancements based on diversity and redundancy can be distinguished: first at the architecture level, where replicas of critical sub-systems are introduced to maintain service delivery even when one sub-system fails; second at the code level, where some program transformations are made to diversify replica; and finally a fully monitored combination of diversified components cleverly assembled in a secure architecture. In the sequel, these approaches are discussed further.

3. Redundancy Obtained by Multiple Instances Running in Parallel

Two categories of software architectures implementing redundancy can be distinguished. The first category uses a proxy to coordinate multiple COTS applications while the second one uses a middleware to achieve the same purpose. Noticeably, some commercial products implementing such strategies can now be found on the market like everRun for Windows by Marathon Technologies.

3.1 Multiple COTS Applications Coordinated by a Proxy

The software architectures described in this section implement the architectural pattern depicted in Figure 1. This approach is ideal for a system integration of COTS components or legacy and closed applications aiming to deliver the services. The servers are shielded from the user side through proxies. Monitoring and voting mechanisms are used to check the health of the system, validate the results, and detect abnormal behavior. Examples of this approach include the Dependable Intrusion Tolerance architecture [11, 12], the Scalable Intrusion Tolerant Architecture [13], and Hierarchical Adaptive Control for QoS Intrusion Tolerance (HACQIT) [14].

3.2 Multiple Applications Assembled Through Middleware

Middleware-based approaches are much richer since they can provide server coordination between multiple “diverse” applications while hiding the sub-system differences [15]. Several intrusion tolerant software architectures are part of this category.

The Intrusion Tolerance by Unpredictable Adaptation architecture is a distributed object framework that integrates several mechanisms to enable the defense of critical applications [16]. The objective of this architecture is to enable the tolerance of sophisticated attacks aimed at corrupting a system.

Malicious and Accidental Fault Tolerance for Internet Applications [17] is a European research project that targeted the objective of systematically investigating the tolerance paradigm in order to build large-scale dependable distributed applications.

The Designing Protection and Adaptation Into a Survivability Architecture [18, 19] is a survivability architecture providing a diverse set of defense mechanisms. This architecture diversity is used to achieve a defense in depth and a multi-layer security approach [19]. This architecture relies on a robust network

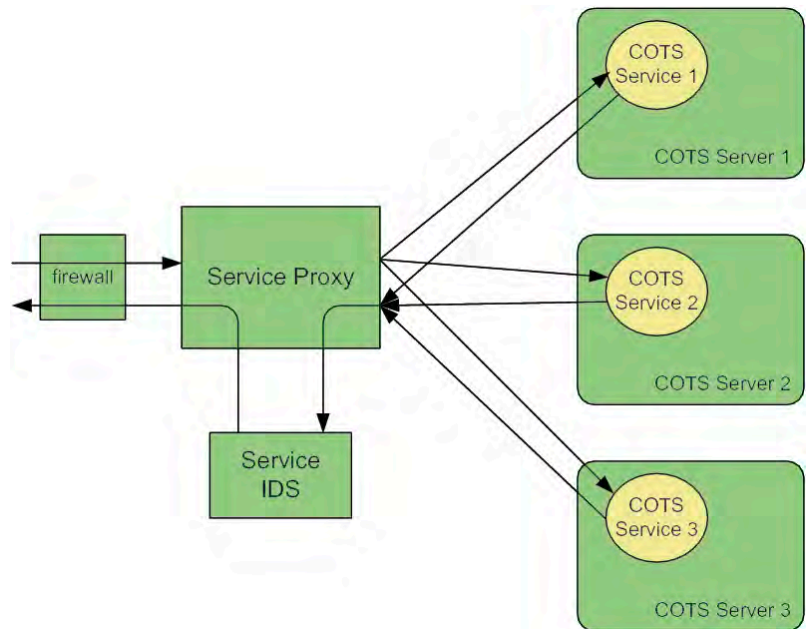


Figure 1: General Architectural Pattern of Intrusion Tolerance

infrastructure that supports redundancy and provides security services such as packet filtering, source authentication, link-level encryption, and network anomaly sensors. The detection of violations “triggers” defensive responses provided by middleware components in the architecture.

Fault/intrusion REmoVal through Evolution and Recovery (FOREVER) [20] is a service that is used to enhance the resilience of intrusion-tolerant replicated systems. FOREVER achieves this goal through the combination of recovery and evolution. FOREVER allows a system to recover from malicious attacks or faults using time-triggered or event-triggered periodic recoveries.

4. Diversity Obtained by Program Transformations

Diversity can be introduced in the software ecosystem by applying automatic program transformations, which preserve the functional behavior and the programming language semantics. They consist essentially in randomization of the code, the address space layout or both in order to provide a probabilistic defense against unknown threats. Three main techniques can be used to randomize software:

Instruction Set Randomization (ISR) [21, 22] changes the instruction set of the processor so that unauthorized code will not run successfully. The main idea underlying ISR is to decrease the attacker’s knowledge about the language used by the runtime environment on which the target application runs. ISR techniques aim at defending against code injection attacks, which consist of introducing executable code within the address space of a target process, and then passing the control to the injected code. Code injection attacks can succeed when the injected code is compatible with the execution environment.

Address Space Randomization (ASR) [23] is used to increase software resistance to memory corruption attacks. These are designed to exploit memory manipulation vulnerabilities such as stack and heap overflows and underflows, format string vulnerabilities, array index overflows, and uninitialized variables. ASR consists basically of randomizing the different regions of the

process address space such as the stack and the heap. Noticeably, ASR has been integrated into the default configuration of the Windows Vista operating system [24].

Data Space Randomization (DSR) is a different randomization-based approach which aims also at defending against memory error exploits [25]. In particular, DSR randomizes the representation of data objects. This is often implemented by applying a modification to the data representation, such as using an Exclusive Or operation for each data object in memory against randomly chosen mask values. The data are unmasked right before being used. This makes the results of using the corrupted data highly unpredictable. The DSR technique seems to have advantages over ASR, as it provides a broader range of randomization (on 32-bit architectures, integers and pointers are randomized over a range of 2^{32} values). In addition, DSR is able to randomize the relative distance between two data objects, addressing a weakness of the ASR technique.

5. Higher Resistance Obtained by Combining Redundancy and Diversity

The ability to build a system combining redundant and diverse components provides new powerful capabilities. One of them is the advanced monitoring of the redundant system by comparing the behavior of the diverse replicas. This endows the system with efficient intrusion detection capabilities not achievable with standard intrusion detection techniques based on signatures or malware modeling. Moreover, with the introduction of some assessment of the behavioral advantages of one implementation over the others, a "meta-controller" can ultimately adapt the system behavior or its structure over time. These futurist concepts are now prototyped in several projects like those briefly described below.

5.1 Intrusion Detection using Output Voting

Several experimental systems used output voting for the sake of detecting some types of server compromise. For example, the HACQIT system [11] uses the status codes of the server replica responses. If the status codes are different the system detects a failure. Totel et al. [26] extend this work to do a more detailed comparison of the replica responses. They realized that web server responses may be slightly different even when there is no attack, and proposed a detection algorithm to detect intrusions with a higher accuracy (lower false alarm rate). These research initiatives specifically target web servers and analyze only server responses. Consequently, they cannot consistently detect compromised replicas.

5.2 Behavior Monitoring in N-Variant Systems

N-variant systems provide a framework that allows executing a set of automatically diversified variants using the same inputs [27]. The framework monitors the behavior of the variants in order to detect divergences. The variants are built so that an anticipated type of exploit can succeed on only one variant. Therefore, such exploits become detectable. Building the variants requires a special compiler or a binary rewriter. Moreover, this framework detects only anticipated types of exploits, against which the replicas are diversified.

5.3 Multi-variant Execution Environment

Multi-variant code execution is a runtime monitoring technique that prevents malicious code execution [28]. This technique uses diversity to protect against malicious code injection attacks. This is achieved by running several slightly different variants of the same program in lockstep. The behavior of the variants is compared at synchronization points, which are in general system calls. Any divergence in behavior is suggestive of an anomaly and raises an alarm.

5.4 Behavioral Distance

The behavioral distance approach aims at detecting sophisticated attacks which manage to emulate the original system behavior including returning the correct service response (also known as mimicry attacks). These attacks are thus able to defeat traditional anomaly-based intrusion detection systems. Behavioral distance achieves this defense using a comparison between the behaviors of two diverse processes running the same input. It measures the extent to which the two processes behave differently. Gao et al. proposed two approaches to compute such measures [29, 30].

6. Concluding Remarks

A few modern operating systems integrate some level of diversity to improve internal security and a few COTS packages are emerging that implement redundancy extension into traditional architectures. It seems that system architects should now consider more systematically redundancy or component diversity for critical systems that are operated in hostile environments. In many instances, the cost of security failures may well justify the additional complexity and the associated deployment and operating costs. The exploitation of both features simultaneously remains mostly experimental at this time but the very strong promises that such architectures make will continue to justify research and development in this field.

ABOUT THE AUTHORS



Dr. Abdelouahed Gherbi is an assistant professor at the École de technologie supérieure, Québec. In 2009-10, he was a visiting researcher at the Defence Research and Development Canada, Valcartier. He completed his Ph.D. in computer engineering at Concordia University in 2008. His main research interests include the modeling and analysis of embedded and real time software as well as the reliability, security and availability of systems.

École de technologie supérieure
1100, rue Notre-Dame Ouest
Montréal (Qc) H3C 1K3
Canada
Phone: 514-396-8465
Fax: 514-396-8405
E-mail: Abdelouahed.Gherbi@etsmtl.ca

ABOUT THE AUTHORS



Mario Couture received a B.Sc. degree in Physics in 1986, a M.Sc. in Physical Oceanography in 1989, and a second M.Sc. in Electrical Engineering at Laval University in 2002. After eight years of work in modeling and simulation at Fisheries and Ocean Canada, he joined Defence R&D Canada. His research interests are mainly oriented the live cyber-surveillance and protection of military information systems against stealth attacks.

Defence R&D Canada - DRDC-RDDC Valcartier
2459 Pie XI Blvd North
Québec, QC, G3J-1X5
Canada
Phone: 418-844-4000 x4285
Fax: 418-844-4538
E-mail : Mario.Couture@Forces.gc.ca



Robert Charpentier completed his degree in engineering physics at École Polytechnique de Montréal in 1979. After working at CAE Electronics on flight simulators, he joined Defence Research Establishment Valcartier, where he specialized in infrared imagery and space-based surveillance. His current research domains are secure interoperability, automated software hardening and software security design.

Defence R&D Canada - DRDC-RDDC Valcartier
2459 Pie XI Blvd North
Québec, QC, G3J-1X5
Canada
Phone: 418-844-4000 x4371
Fax: 418-844-4538
E-mail: Robert.Charpentier@Forces.gc.ca

REFERENCES

1. Symantec (2009), Global Internet Security Threat Report – Trends for 2008, (TR XIV), Symantec.
2. Emerging risks team, Lloyds (2009), Digital Risks: Views of a Changing Risk Landscape, (TR XIV) Lloyd's.
3. Lala, Jaynarayan H.; and Schneider, Fred B. (2009), IT Monoculture Security Risks and Defenses, IEEE Security & Privacy, 7(1), pp. 12–13.
4. Birman, K.P.; Schneider, F.B. (2009), "The Monoculture Risk Put into Context," IEEE Security & Privacy, vol.7, no.1, pp.14–17
5. Avizienis, Algirdas, Laprie, Jean-Claude, Randell, Brian, and Landwehr, Carl E. (2004), Basic Concepts and Taxonomy of Dependable and Secure Computing, IEEE Trans. Dependable Sec. Comput., 1(1), 11–33.
6. Forrest, Stephanie; Somayaji, Anil; and Ackley, David H. (1997), Building Diverse Computer Systems, Workshop on Hot Topics in Operating Systems, pp. 67–72.
7. Deswarte, Yves; Kanoun, Karama; and Laprie, Jean-Claude (1998), Diversity against accidental and deliberate faults, In Computer Security, Dependability, and Assurance: From Needs to Solutions, IEEE Computer Society Press, pp. 171–181.
8. Obelheiro, Rafael R.; Bessani, Alysson N.; Lung, Lau C.; and Correia, Miguel (2006), How Practical are Intrusion-Tolerant Distributed Systems? (TR0615) Universidade de Lisboa.
9. Bain, Charles; Faatz, Donald B.; Fayad, Amgad; and Williams, Douglas E. (2001), Diversity as a defense strategy in information systems. In IFIP Proceedings, Vol. 211, Kluwer, pp. 77–94.

REFERENCES

10. Ammann, Paul; Barnes, Bruce H.; Jajodia, Sushil; and Sibley, Edgar H. (Eds.) (1999), Computer Security, Dependability, and Assurance: From Needs to Solutions: Proceedings 7–9 July 1998, York, England, 11–13 November 1998, Williamsburg, Virginia, IEEE Computer Society Press.
11. Deswarte, Yves; and Powell, David (2004), Intrusion tolerance for Internet applications, In Proceedings of the IFIP 18th World Computer Congress, August 22–27, 2004, Toulouse, France, Kluwer, pp. 241–256.
12. Valdes, Alfonso; Almgren, Magnus; Cheung, Steven; Deswarte, Yves; Dutertre, Bruno; Levy, Joshua; Saidi, Hassen; Stavridou, Victoria; and Uribe, Tomas E. (2003), Dependable Intrusion Tolerance: Technology Demo, In Proceedings of the 3rd DARPA Information Survivability Conference and Exposition, April 22–24, 2003, Washington, DC (DISCEX-III 2003), IEEE, pp. 128–130.
13. Wang, Feiyi; Jou, Frank; Gong, Fengmin; Sargor, Chandramouli; Goseva-Popstojanova, Katerina; and Trivedi, Kishor (2003), SITAR: A Scalable Intrusion-Tolerant Architecture for Distributed Services, In Proceedings of the Foundations of Intrusion Tolerant Systems (OASIS'03), IEEE, pp. 359–367.
14. Reynolds, James C.; Just, James E.; Lawson, Ed; Clough, Larry A.; Maglich, Ryan; and Levitt, Karl N. (2002), The Design and Implementation of an Intrusion Tolerant System, In Proceedings of the 2002 International Conference on Dependable Systems and Networks (DSN 2002), IEEE, pp. 285–292.
15. Sarnes, David; Matt, Brian; Niebuhr, Brian; Tally, Gregg; Whitmore, Brent; and Bakken, David E. (2002), Developing a Heterogeneous Intrusion Tolerant CORBA System, In Proceedings of the 2002 International Conference on Dependable Systems and Networks (DSN 2002), IEEE, pp. 239–248.
16. Pal, Partha Pratim; Rubel, Paul et al. (2006), An architecture for adaptive intrusion-tolerant applications, Software: Practice and Experience, 36(11–12), pp. 1331–1354.
17. Verissimo, Paulo; Neves, Nuno Ferreira; Cachin, Christian; Poritz, Jonathan A.; Powell, David; Deswarte, Yves; Stroud, Robert J.; and Welch, Ian (2006), Intrusion-tolerant middleware: the road to automatic security, IEEE Security & Privacy, 4(4), pp. 54–62.
18. Atighetchi, Michael; Rubel, Paul; Pal, Partha Pratim; Chong, Jennifer; and Sudin, Lyle (2005), Networking Aspects in the DPASA Survivability Architecture: An Experience Report, 4th IEEE International Symposium on Network Computing and Applications, IEEE, pp. 219–222.
19. Chong, Jennifer; Pal, Partha Pratim; Atighetchi, Michael; Rubel, Paul; and Webber, Franklin (2005), Survivability Architecture of a Mission Critical System: The DPASA Example, In Proceedings of the 21st Annual Computer Security Applications Conference, IEEE, pp. 495–504.
20. Bessani, Alysson Neves; Reiser, Hans P.; Sousa, Paulo; Gashi, Ilir; Stankovic, Vladimir; Distler, Tobias; Kapitza, Rüdiger; Daidone, Alessandro; and Obelheiro, Rafael R. (2008), FOREVER: Fault/Intrusion Removal through Evolution & Recovery, ACM/IFIP/USENIX 9th International Middleware Conference, Association for Computing Machinery (ACM), pp. 99–101.
21. Kc, Gaurav S.; Keromytis, Angelos D.; and Prevelakis, Vassilis (2003), Countering code-injection attacks with instruction-set randomization, Proceedings of the 10th ACM Conference on Computer and Communications Security, ACM, pp. 272–280.
22. Barrantes, Elena Gabriela; Ackley, David H.; Palmer, Trek S.; Stefanovic, Darko; and Zovi, Dino Dai (2003), Randomized instruction set emulation to disrupt binary code injection attacks, Proceedings of the 10th ACM Conference on Computer and Communications Security, ACM, pp. 281–289.
23. Shacham, Hovav; Page, Matthew; Pfaff, Ben; Goh, Eu-Jin; Modadugu, Nagendra; and Boneh, Dan (2004), On the effectiveness of address-space randomization, In Proceedings of the 11th ACM Conference on Computer and Communications Security, ACM, pp. 298–307.
24. Whitehouse, Ollie (2007), An Analysis of Address Space Layout Randomization on Windows Vista, Technical Report Symantec.
25. Bhatkar, Sandeep and Sekar, R. (2008), Data Space Randomization, Vol. 5137 of Lecture Notes in Computer Science (LNCS): 5th Conference on Detection of Intrusions and Malware & Vulnerability Assessment, July 10–11, 2008, Paris, France (DIMVA 2008), Springer, pp. 1–22.
26. Totel, Eric; Majorczyk, Frédéric; and Mé, Ludovic (2006), COTS Diversity Based Intrusion Detection and Application to Web Servers, In LNCS Vol. 3858: 8th International Symposium on Recent Advances in Intrusion Detection, September 7–9, 2005, Seattle, Washington (RAID 2005), Springer, pp. 43–62.
27. Cox, Benjamin; Evans, David; Filipi, Adrian; Rowanhill, Jonathan; Hu, Wei; Davidson, Jack; Knight, John; Nguyen-Tuong, Anh; and Hiser, Jason (2006), N-variant systems: a secretless framework for security through diversity, In Proceedings of the 15th conference on USENIX Security Symposium, USENIX Association.
28. Weatherwax, Eric; Knight, John; and Nguyen-Tuong, Anh (2009), A Model of Secretless Security in N-Variant Systems, In Proceedings of the 39th Annual IFIP Conference on Dependable Systems and Network (DSN 2009).
29. Gao, Debin; Reiter, Michael K.; and Song, Dawn Xiaodong (2006), Behavioral Distance for Intrusion Detection, LNCS Vol. 3858: 8th International Symposium on Recent Advances in Intrusion Detection, September 7–9, 2005, Seattle, Washington (RAID 2005), Springer, pp. 63–81.
30. Gao, Debin; Reiter, Michael K.; and Song, Dawn Xiaodong (2006), Behavioral Distance Measurement Using Hidden Markov Models, RAID'06, LNCS Vol. 4219: 9th International Symposium on Recent Advances in Intrusion Detection, September 20–22, 2006, Hamburg, Germany (RAID 2006), Springer, pp. 19–40.

Creating Attack-Aware Software Applications with Real-Time Defenses

Colin Watson, OWASP
Michael Coates, OWASP
John Melton, OWASP
Dennis Groves, OWASP

Abstract. Attack-aware software applications provide attack detection and real-time defensive response with a very low false-positive rate. This technique allows an application to detect and neutralize a threat before the attacker exploits a known or unknown vulnerability. The approach is especially suited to software applications with high information assurance requirements such as in the defense, critical national infrastructure, and financial service sectors to protect against cyber espionage, fraud, business logic abuse, tampering, and theft. The Open Web Application Security Project (OWASP) has developed a methodology, documentation, code and pilot demonstration which can be freely used to apply the concepts; this project is called AppSensor.

Introduction

Information systems and data are being targeted relentlessly by skilled and motivated adversaries who are well resourced and have excellent tools. These attackers, who may be backed by organized crime, governments, or commercial enterprises, identify and exploit vulnerabilities in applications themselves to access sensitive and secret data.

In this article we discuss why conventional application defenses are not a solution to these types of advanced attacks, and explain an initiative from OWASP [1] that utilizes self-defense to provide real-time detection and response.

Conventional Defensive Measures

A fundamental starting point for secure applications is during the development lifecycle. The goal is robust code—designed, developed, configured and operated on hardened operating systems. However, this does not mean the software is impregnable; unknown vulnerabilities almost certainly exist, new vulnerabilities are introduced with software code changes and attackers examine and probe applications to find them. Once they identify one or more vulnerabilities, they discover ways to exploit them. It is very hard to detect these custom attacks using conventional defensive measures.

Some techniques often thought to defend applications include using Transport Layer Security (TLS), network firewalls, application gateways (e.g., web application firewalls, XML appliances, cross-domain guards), and network/host Intrusion Detection and Prevention Systems (IDPS).

TLS provides confidentiality and integrity. This prevents attackers from eavesdropping on traffic but attackers' payloads are sent to the server in the same way as normal traffic. Since TLS simply encrypts attackers' malicious data as it is sent to the server, it provides no protection from these attackers.

Network firewalls are a vital component for network defense, but they allow or deny traffic over a particular port. By necessity, web applications need to allow traffic through specified ports and attackers send their payloads using the same ports. Application gateways, even with careful configuration and self-learning, are also generic solutions to typical problems and IDPS has no real concept of good and bad application usage.

Additionally, some of these conventional defenses try to replicate part of the application logic but they can, at best, only do this partially. This external logic then needs to be verified, maintained, protected and managed, adding another often significant overhead to operational costs.

The generality of conventional defensive measures does mean they can be applied to numerous applications, often with little or no tuning, with obvious benefits such as cost and time to deploy. However, a significant challenge for this strategy is a lack of context. Detection of known attacks will almost certainly fail to detect a custom attack looking to exploit a weakness within an application. A different approach is required as conventional defenses are not working.

Application Defensive Measures

The right way to detect advanced attacks is within the applications themselves, by adding application-specific security controls that detect malicious activity.

Consider the analogy of control instrumentation in an industrial production process. Measurement points for temperature, mass, velocity, etc. are added to quantify key aspects in real-time. This information may be used in localized control, but is usually integrated into an overall process control system, which aggregates signals from many sensors, and uses various control mechanisms to react to changes and maintain a desired state. Unfortunately, most current software applications are like industrial processes without control instrumentation—there are no sensors and no intelligence about what is occurring—they are blind to what is happening internally.

But unlike conventional defensive measures, software applications have full knowledge about the business logic and the roles and permissions of users—each application can make informed decisions about misuse, and identify and stop attackers with a very low false-positive rate. An additional benefit is that this context-aware analysis occurs in real-time. This is an application-specific approach, not a generic one, because the controls are related and integrated within the application, and can be selected based on an assessment of risks specific to the process and data.

Similar ideas already exist in some software, but these are usually implemented as isolated processes and some may be undertaken reactively to events, or performed largely in a manual way. The concept described in this article focuses and formalizes this approach. It is about implementing proactive measures to add instrumentation and controls directly into an application in advance so that events are centrally analyzed and responded to.

Attack-Aware Detection

Once sensors have been deployed within the application code, they transmit detected event data to the analysis engine. This is then responsible for evaluation of individual events and determination of an attack. Once an attack has been recognized, the engine selects and executes an appropriate response.

Software applications then become attack-aware and it is possible to undertake proactive defense against currently unknown threats. Applications are the appropriate place to detect many types of attacks given the wealth of information available to them.

Normal and Malicious Behavior

The approach relies on being able to differentiate between normal and malicious behavior. That is one of the biggest hurdles when security controls (attack detection mechanisms) are added outside the application; it is difficult to distinguish user intent and this leads to a higher rate of false positives.

The ability to differentiate between normal behavior, suspicious behavior and attacks (and to do it with a high degree of granularity) can also be an improvement in the usability of human interfaces of the software applications. Application developers know which inputs allow keyboard entry, which have client-side validation, which should not be altered by a user and which are from trusted systems.

Evasion and Unknown Attacks

A common attack technique is using obfuscation to bypass black list inspection performed by security controls outside of the application. Integrating attack detection within the application solves this problem since all data is fully canonicalised. At this point it is simple for the application to inspect the user-submitted data and determine if it is malicious because the data is no longer obfuscated.

Detection within the application also allows insight into unknown attacks. Detection points can be designed to capture activity that could only occur outside the normal flow of the application. For example, a user would never accidentally trigger a detection point monitoring the use of the HTTP POST method where only the GET method is expected. In this example, the detection point can provide alerts that most likely represent attackers probing for weaknesses within the application.

OWASP AppSensor Project

OWASP AppSensor was developed under the OWASP Season of Code 2008 [2] by Michael Coates. The result is a ground-breaking work defining the AppSensor [3] concept, available as a printed book [4] and as a free PDF download [5]. Since its initial publication, a reference implementation in Java

Detection Point Type	Code	Name
Signature	RE	Request Exceptions
	AE	Authentication Exceptions
	SE	Session Exceptions
	ACE	Access Control Exceptions
	IE	Input Exceptions
	EE	Encoding Exceptions
	CIE	Command Injection Exceptions
	FIO	File IO Exceptions
	HT	Honey Trap
Behavioral	UTE	User Trend Exceptions
	STE	System Trend Exceptions
	RP	Reputation

Table 1: AppSensor guidance lists over 50 example detection point definitions grouped into these 12 categories; additional custom detection points are also often required.

[6], a demonstration pilot [7], and several presentations and videos [8] have been developed.

Like an industrial process control system instrumentation engineer, developers have a whole range of sensors available. Most application sensors are located within the application code itself, but AppSensor also has the notion of inputs from other "external" devices. As an example, the software could alter its security posture based on external threat ratings such as the United States Armed Forces Defense Readiness Condition. AppSensor defines over 50 detection points [9] (Table 1) for monitoring. The list is not definitive, but is a starting point as most applications will need to consider adding custom detection points at various points deemed important.

AppSensor works best within the authenticated portion of an application where the user's identity is known and can be blocked if malicious, but it is also possible to apply the principles to other areas. Although the concepts can be retrofitted to existing software, they are most-easily built in during software development and the AppSensor document has guidance on detection point considerations, determining the malicious intent, monitoring system trend events and implementation guidance. However, the planning stages are probably the most time-consuming aspect of implementing AppSensor. If threat modeling is already being used in your software development lifecycle, this may be a natural location to determine many of your application specific detection points. Additional guidance [10] is available for assistance with this stage and includes a step-by-step description of how to plan implementation. Adoption can be encouraged, and the benefits realized by building AppSensor requirements into project specifications. Retrofitting existing software can also be undertaken, by building separate code libraries and calling these from within existing routines. The OWASP Enterprise Security API (ESAPI) [11] security control library can be used directly, or as a model for custom code library development.

Response Type	Examples
Allow	Log events only
Deny	Block IP address

Table 2: Conventional defensive measures may only offer a binary choice such as logging or blocking.

Rich Responses

Once the detection points have been determined, thresholds are set for each point, or group of points, which then act like tripwires. In AppSensor, an analysis engine monitors the detection events and determines when and what response is appropriate according to these specified event thresholds. Conventional defenses offer only binary response options such as allow/deny or log/block (Table 2). They are often limited to a single dimension—the particular network connection contravening a rule.

The AppSensor concept describes a wide spectrum of possible responses [12], and when combined with the very low false-positive detection rate, defenses do not need to be configured in detection-only mode for fear of preventing valid traffic. This provides a two-dimensional view, where responses may be directed not only against a single connection or a single user, but now also against a whole group (e.g., based on location, role, or behavior), or against all users. Table 3 lists some common types of response.

When behavioral aspects over time (a request/response, a session, a time period) are included, the application becomes capable of rich three-dimensional responses. The behavioral aspects can span beyond application restarts, upgrades and even the application lifecycle from deployment to disposal, since data from other systems that pre-date the application may be incorporated.

Cross Integration

As mentioned above, AppSensor can make use of information from other systems and devices to contribute to its pool of information for attack detection, to provide greater value. External systems may also be able to contribute to AppSensor's

Response Type	Examples
Logging Change	Full stack trace of error messages logged Record DNS data on user's IP address
Administrator Notification	Visual indicator displayed on an application monitoring dashboard Audible alarm in the control room
Other Notification	Signal sent to upstream network firewall, application firewall (e.g. XML, web) or load balancer Alert sent to fraud protection department
Proxy	Requests from the user invisibly (from the user's perspective) passed through to a hardened system Request are proxied to a special honeypot system which closely mimics or has identical user functionality
User Status Change	Change Increase data validation strictness for all form submissions by this citizen Reduce the number of failed authentication attempts allowed before the user's account is locked
User Notification	On-screen message about data validation issues Message sent by email to the registered email address to inform them their password has been changed
Timing Change	File upload process duration extended artificially Add fixed time delay into every response
Process Terminated	Discard data, display message and force user to begin business process from start Redirection of an unauthenticated user to the log-in page
Function Amended	Limit on feature usage rate imposed Additional validation requirements
Function Disabled	Web service inactivated Content syndication stopped
Account Logout	Session terminated and user redirected to logged-out message page Session terminated only (no redirect)
Account Lockout	User account locked permanently until an Administrator resets it One user's IP address range blocked
Application Disabled	Website shut down and replaced with temporary static page Application taken offline
Collect Data from User	Deploy a Java applet to collect remote IP address Deploy JavaScript to collect information about the user's network

Table 3: AppSensor provides the ability to have a much richer range of responses; the guidance lists 14 types of response. Specific examples for each type show how the types can be customized for each application's context.

decision-making processes. For example behavioral information can be fed into stochastic systems for statistical analysis, which may be the single most important pattern against unknown attacks to date. This defense against the unknown attacks is a giant benefit that can not be achieved using anything else, anywhere, at any price.

As well as consuming data, AppSensor can pass back intelligence to other systems. This might be used as part of the real-time response:

- Locking a user's single-sign-on account
- Removing some permissions for a user from a physical access control system
- Setting a warning flag about a customer
- Blocking a session ID or IP address at a network firewall.

Equally, data can be federated to other systems such as Security Information and Event Management (SIEM), Intrusion Detection Systems and fraud monitoring systems. One adopter in particular uses AppSensor heavily integrated with SIEM.

Information Insight and Additional Value

AppSensor provides insight into whether, and how, attacks are occurring. A pilot implementation has also demonstrated the value this approach has to containing the damage caused by application worms. In production applications, system trend detection points have been used to identify changes in a function's usage, alert administrators and ultimately disable the function. While this does not remove an infection, it stops a worm from spreading, limits the extent of data requiring clean-up, and may allow the remainder of the application to continue functioning, which could otherwise have had to be shut down. Another benefit is that AppSensor provides useful security metrics, currently in absentia.

Applicability

Software applications must be built securely in the first place. If issues like authentication, session management, authorization, etc. are broken already, it will not be possible to implement the approach. Self-defense goes beyond the implementation of security services, but there does appear to be growing interest in this area. For example, the DoD recently announced [13] that it is planning to spend \$500 million to research new cyber security technologies including "active defenses"—technologies that detect attacks and probes as they occur.

AppSensor is a comprehensive proactive, rather than reactive, approach that can be applied to applications in many situations. It reduces the risk of unknown vulnerabilities being exploited by identifying and containing users conducting malicious activity that is often the precursor to an attack. It greatly increases the visibility of suspicious events and actual attacks. This can provide additional information assurance benefits:

- Reduced security risks to data and information systems
- Improved compliance
- Reduction in the consequences of data breaches.

In turn, these can provide improved service levels and resilience that have wide applicability. Some types of organizations may also achieve competitive advantage.

Although these concepts are already deployed in production environments, the AppSensor team is considering ways to improve the analysis engine including the use of stochastic methods, increasing coverage in ESAPI, integration with other systems and building the concepts into software frameworks and libraries so the implementation overhead is reduced. The team would also like to investigate how AppSensor could be applied in an agile/test-driven development environment.

Conclusions

All types of organizations have very powerful systems with access to mission-critical data, but generally have no idea if, when or how applications are under attack. This is the crux of the problem: critical data on critical systems and little to no visibility of the health of the overall environment. This must change. The concepts embodied by OWASP AppSensor present a solution to this issue. By creating applications that are attack-aware and able to respond appropriately, the assurance of information systems is increased.

Attack detection and response actions can be integrated into any application to greatly increase the overall security of the application. At a minimum, this technology provides greatly needed insight into the current attacks and malicious activities occurring within the application. When properly tuned, the attack-aware application will be able to identify and defend against malicious users in real-time. This technology must be implemented into mission-critical applications as we continue to see sensitive data and controls moving to systems that are subjected to a constant stream of malicious attacks.

Copyright

The OWASP Foundation

Acknowledgements

The authors would like to acknowledge the support and collaboration of everyone who has contributed their ideas and time to the AppSensor project, and to the OWASP Foundation for providing the initial Summer of Code funding in 2008. ♦

ABOUT THE AUTHORS



Colin Watson's work involves the management of application risk, building security and privacy into software development processes and keeping abreast of international legislation and standards. He holds a BSc in Chemical Engineering from Heriot-Watt University in Edinburgh, and an MSc in Computation from the University of Oxford. He contributes to a number of OWASP projects and is a member of the OWASP Global Industry Committee. Colin is a consultant and co-founder of Watson Hall Ltd.

E-mail: colin.watson@owasp.org

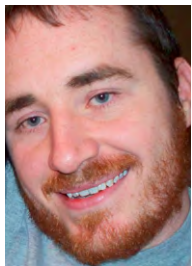
Continued on next page.

ABOUT THE AUTHORS



Michael Coates has extensive experience in application security, security code review and penetration assessments. He holds an MSc in Computer Security from DePaul University and a BSc in Computer Science from the University of Illinois. Michael is the creator and leader of the AppSensor project, a contributor to the OWASP Top 10 and a frequent speaker at security conferences. He is in charge of Mozilla's Infrastructure Security team responsible for the security lifecycle of web applications, and the protection of infrastructure and sensitive user data.

E-mail michael.coates@owasp.org



John Melton is a software architect and designer with particular knowledge of security for online systems. He is the AppSensor project's development lead, and has been responsible for most of the example code developed. He holds an MSc in Information Security and a BSc in Computer Science, both from the University of North Carolina at Charlotte. John is a Senior Information Security Engineer at Wells Fargo and lectures at UNC Charlotte.

E-mail john.melton@owasp.org



Dennis Groves is the co-founder of OWASP. He is a well-known thought leader in application security whose work focuses on multidisciplinary approaches to information security risk management. He holds an MSc in Information Security from the University of Royal Holloway, University of London. He is currently an expert for the UK mirror of ISO subcommittee 27, WG4.

E-mail dennis.groves@owasp.org

The OWASP Foundation
9175 Guilford Road Suite #300
Columbia, MD 21046
Telephone: 301-275-9403
Fax: 301-604-8033

REFERENCES

1. The Open Web Application Security Project (OWASP). About the Open Web Application Security Project. <http://www.owasp.org/index.php/About_OWASP>
2. Ibid. Summer of Code 2008. <http://www.owasp.org/index.php/OWASP_Summer_of_Code_2008>
3. Ibid. AppSensor Project. <http://www.owasp.org/index.php/Category:OWASP_AppSensor_Project>.
4. Coates, Michael. AppSensor - Detect and Respond to Attacks from Within the Application, v1.1, OWASP. Paperback, 48 pages, Lulu. <<http://www.lulu.com/product/paperback/owasp-appsensor/4520003>>
5. Ibid. AppSensor - Detect and Respond to Attacks from Within the Application, v1.1, OWASP. <https://www.owasp.org/images/2/2f/OWASP_AppSensor_Beta_1.1.pdf>
6. OWASP. AppSensor Developer Guide. <http://www.owasp.org/index.php/AppSensor_Developer_Guide>
7. Ibid. AppSensor Live Tutorial. <<http://www.defendtheapp.com/>>
8. Ibid. AppSensor Media. <http://www.owasp.org/index.php/OWASP_AppSensor_Project#tab=Media>
9. Ibid. AppSensor Detection Points. <http://www.owasp.org/index.php/AppSensor_DetectionPoints>
10. Watson, Colin. AppSensor Implementation Planning Workbook, OWASP, December 2010. <<http://www.owasp.org/index.php/File:Appsensor-planning.zip>>
11. OWASP. Enterprise Security API. <http://www.owasp.org/index.php/Category:OWASP_Enterprise_Security_API>
12. OWASP. AppSensor Response Actions. <http://www.owasp.org/index.php/AppSensor_ResponseActions>
13. Ratnam, Gopal and King, Rachael. "Pentagon Seeks \$500 Million for Cyber Technologies", Bloomberg, 15 February 2011. <<http://www.bloomberg.com/news/2011-02-15/pentagon-seeks-500-million-for-cyber-research-cloud-computing.html>>

CALL FOR ARTICLES

If your experience or research has produced information that could be useful to others, **CROSSTALK** can get the word out. We are specifically looking for articles on software-related topics to supplement upcoming theme issues. Below is the submittal schedule for three areas of emphasis we are looking for:

Securing a Mobile World

March/April 2012 Issue

Submission Deadline: Oct 10, 2011

Rapid and Agile Stability

May/June 2012 Issue

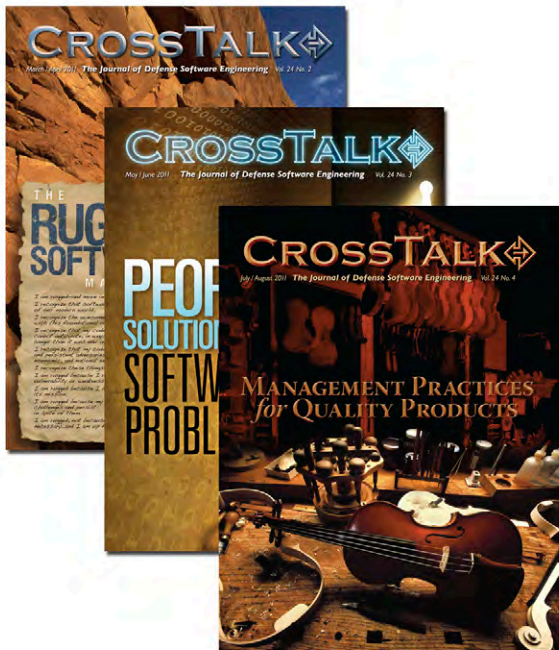
Submission Deadline: Dec 10, 2011

The End of the PC

July/Aug 2012 Issue

Submission Deadline: Feb 10, 2012

Please follow the Author Guidelines for **CROSSTALK**, available on the Internet at <www.crosstalkonline.org/submission-guidelines>. We accept article submissions on software-related topics at any time, along with Letters to the Editor and BackTalk. To see a list of themes for upcoming issues or to learn more about the types of articles we're looking for visit <www.crosstalkonline.org/theme-calendar>.



Creating Data from Applications for Detecting Stealth Attacks

C. Warren Axelrod, Ph.D., Delta Risk LLC

Abstract. A major reason for security professionals not seeming able to protect fully against the rapidly changing threat environment and sophisticated attacks is that they often don't have available the necessary application security data for detecting and responding to such increasingly stealthy attacks. Furthermore, easily generated application security data are generally not reported timely or accurately enough for appropriate preventative action. In this article, we describe the issues confronting those attempting to create, collect, report, and respond to data—and which are more useful but harder to come by. We also suggest how these impediments might be overcome.

Introduction

Cyber security staffs collect and analyze huge volumes of security-related data. The problem is that applications do not generate the most useful data in the first place [1]. So many major data breaches reportedly occur without the knowledge of their victims. It appears that ChoicePoint [2], Heartland Payments [3], NASDAQ [4], and Epsilon [5], for example, only find out the degree to which data in their custody has been compromised when they are notified by third parties, such as VISA International, or called in forensics specialists, such as the NSA. For example, VISA International might observe concentrations of fraudulent payment card activity and trace them back to the company that had suffered the data leak. Even then, victim companies are frequently not able to determine who did what, and when.

There is ample evidence for the need for such information. According to the Verizon Business 2010 Data Breach Investigations Report [6], a large percentage of total breaches originate from insiders, specifically:

- **48 % of data breaches were caused by insiders**
- **48 % of data breaches involved privilege misuse**

It should be noted that the sample upon which the results of the Verizon Business analysis is based refers to data collected from their own clients, and therefore contains considerable bias. However, the general observations still make sense, even though the data are not necessarily representative of all organizations. A lesson taken from these statistics is that the lack of identification by victim organizations that a breach, particularly an inside job, has occurred leads one to the view that either the data are available but not analyzed (the supposition of the report) or they are not available, to which view the author subscribes. This premise,

namely that the data are not generated unless specific actions are taken to create them, is the basis for this article.

According to a Veriphys white paper [7], the major causes of insider attacks are the granting of excess unneeded privileged access rights, which often results from such practices as not deleting obsolete access rights, and so on. While this clearly contributes to the problem, it is by no means the only cause. Many insider attacks result from valid access rights that allow certain fraudulent activities to proceed undetected. In other cases, the system granting user access does not have fine enough granularity to be able to carve out only those access rights that are needed to perform a particular function.

Scope

Software systems and the data that they access are usually protected at a number of levels. Traditionally we find controls for system access, use, and access termination. Access security services include authentication and authorization, such as are embodied in the DoD's Common Access Card (CAC). The CAC was developed in order to be in compliance with the policy for a common identification standard for federal employees and contractors as described in Homeland Security Presidential Directive 12/HSPD-12 [8]. There are readily available monitoring and reporting products and services that identify and analyze user access logs, as well as high-level system use and data exfiltration logs. The latter typically cover e-mail and file transfer systems. Current products and services use readily available logged data, with few requiring the generation of additional data. This feature of not having to create additional data is often given as a benefit of such systems, and to the extent that they are simpler to install and get running, this may be the case. However, there are limitations in such an approach, not the least of which is the risk that key data may not be available for analysis.

Some Widely Publicized Examples

There are many cases in the government and private sectors where a lack of application-generated data has resulted in major breaches or system failures. An April 2011 attack on Sony PlayStation customer data—where the perpetrators used Amazon Web Services as their point of takeoff—is just such an example. In an article on the case [9], Pete Malcolm, CEO of Abiquo Inc., is quoted as saying, "There is no way of telling who's a good guy and who's a bad guy." An earlier release of the article, which was subsequently replaced with the referenced article, claims that Malcolm said that, "Amazon does not have the means to detect illegal uses of its servers."

A widely publicized example relating to the DoD is that of Private First Class Bradley E. Manning who was charged with unauthorized use and disclosure of U.S. diplomatic cables to WikiLeaks. An overview on Wikipedia [10] states the following: "Manning had been assigned in October 2009 to a support battalion with the 2nd Brigade Combat Team, 10th Mountain Division, based at Forward Operating Base Hammer, near Baghdad. There he had access to the Secret Internet Protocol Router Network, used by the United States government to transmit classified information."

This would appear to be a case in which the perpetrator was authorized to access certain classified information but his use of the information was not flagged as suspect nor brought to the attention of authorities.

In this article, we propose an approach wherein one first determines in advance what additional data should be generated above and beyond the standard data logs. Data collection capabilities are introduced by injecting data requirements throughout the System Development Lifecycle (SDLC). When it comes to COTS and open-source software, where the acquiring entity does not have any control over the software development process, one must request that the vendor or community incorporate the necessary data collection capabilities.

Data Creation

As alluded to above, the author believes that this failure to discover intrusions is largely due to applications not collecting and reporting key information needed to detect hacker activity at the time of an attack or to trace such activities forensically after the fact. This lack of information occurs even when organizations have installed a full complement of monitoring, logging, and Security Information and Event Management (SIEM) tools and have staff diligently reviewing the outputs from these tools.

It is important to ensure that applications now in development, or currently being planned, will incorporate the instrumentation necessary to collect data required to alert security staff that a compromise is taking place or that one has happened. Consequently, application security professionals must be involved in the requirements, specifications, design, testing, deployment and operational phases of the SDLC. Some individual or group, with specific in-depth knowledge of not only an application's functionality but also what security data should be collected, has to be involved in all the above phases of the SDLC. The security data requirements generated by such specialists, though likely to be onerous, must be accepted and incorporated into applications if there is to be any hope of tackling the issue of undetected and undetectable attacks.

Producing the Data

Creating and monitoring relevant data are the keys to success but, as one would expect, the task of defining the full range of required security data is a major challenge since the required multidiscipline expertise rarely exists in the desired quantity and quality. Nevertheless, defining requirements is the right place to start.

Ideally one should collect data for every conceivable user or machine action and misuse that might occur within applications and against databases. This is, of course, an impossible task

and, even were it theoretically attainable, it would be prohibitively expensive. The point is not that we must anticipate every possibility of misuse, but that the right individuals need to be involved in the SDLC process. This will encourage those responsible for protecting information assets to think about what instrumentation needs to be included in order to capture significant misbehavior.

If additional monitoring is implemented, not only does it serve to improve capturing inappropriate and suspicious activities, it also allows test engineers to cover a greater range of possible security gaps since data, which were previously not captured, now become available for testing a fuller range of functionality [11].

Current Data Collection Approach

Many IT shops just collect and report when users gain access to and depart from systems and networks as such events are readily obtained and are easy to specify and understand. They might also collect data about successful and unsuccessful logons in an attempt to uncover those trying to guess the passwords of others. While gathering such information is useful and often mandated by auditors, it has limited value since it does not tell you what authorized and unauthorized users did during their sessions. Also, if generic IDs are permitted and users are allowed to share passwords (both of which practices should be prohibited), there is no way of knowing if the authorized user or a different person is logging on. The user might know this other person or not.

As stated above, the main benefit to be derived from logon/logoff data is that they are easily defined and collected. It also does have some forensics value since one can determine retrospectively who supposedly accessed systems and when. On the other hand, such data are a rich source of audit findings, since auditors can easily cross reference employee lists against user IDs and determine if the user ID was used after the person left the organization.

The Need for a Better Approach

Stepping beyond collecting simple logon/logoff data usually requires a quantum jump in knowledge and effort, which raises questions about return on investment. However, since it is relatively easy for criminals to obtain application access credentials through social engineering means, such as phishing, checking someone at the gate is becoming less effective. Therefore, one is forced to look to other means of ferreting out unauthorized intruders, particularly when they are already inside the application. The sequence of complexity of data collection is shown in Figure 1. Even in this simple example, we see the need to provide more detailed requirements to the development team as we progress in complexity from Level 1 through Level 3.

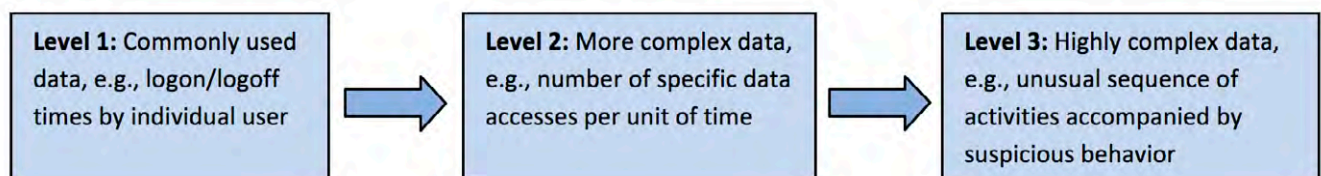


Figure 1: Increasing Complexity of Security Data

Table 1: Event Data Collected by Type of Data

Logon/Logoff Data	User Data	Transaction Data
Type of event (login/logoff) User name Terminal name Timestamps for logons and logoffs Timestamps for department changes Original and new departments Source IP of user Module of application in use Connection serial number Session ID	Type of event (user) Internal ID of user Internal company ID User's company Time when user inserted into database Time when data updated Time when user leaves system User's control number User's account number	Type of event (transaction) Transaction ID ID of user who inserted transaction ID of client requesting service IDs of corporate business group and division Time when transaction inserted into database Time when transaction closed User's company User's control number User's account number

We shall now examine the various levels of complexity according to the difficulty of security data identification, collection, analysis, and limitations on decision-makers' ability to respond to the metrics obtained.

Level 1: Generally Available Data

As indicated above, the easiest data to collect are generally those that are included with most information systems and security tools, such as Identity and Access Management systems. The need to know who logged on and when they did so is a basic requirement for auditing purposes. Even at this level, however, reports showing which applications a person may access are often inaccurate, out of date and difficult to interpret, according to the author's experience with many such reports. Also, the information is often available too late to be able to mitigate the impact of breaches.

Level 2: More Difficult to Identify, Create, Collect and Understand

In the moderately complex category, some data may be collected directly or calculated from other data. For example, the logon/logoff files can be analyzed to show the number of accesses by person by time of day, day of week, etc. Some systems have built-in capabilities to drill down further. For example, special-purpose e-mail monitoring systems provide the following:

- **Records of senders and recipients of e-mails**
- **Whether attachments were included**
- **Whether specific personal and other sensitive information was included in the body of the e-mail or in attachments, and**
- **What that specific information was.**

On the other hand, general business applications, particularly those that have been custom-built, are unlikely to have such capabilities.

In order to achieve this next level of data collection and analysis, it is often necessary to set up a specific initiative as part of the SDLC effort to add event monitoring, data collection, storage, analysis and reporting capabilities. This can typically take several man-months of effort, depending on the size and complexity of the systems. The data items in Table 1 were ob-

tained through just such a project conducted by two companies and supported by a SIEM system vendor. As the table entries show, with a little effort, one can achieve much greater insight as to activities occurring within critical systems.

Level 3: Very Difficult to Identify, Create, Collect and Analyze

As we progress along the difficulty-complexity spectrum, we arrive at a level where a major difficulty is to determine what should be collected, and then the challenge is to come up with ways of creating, collecting, analyzing, reporting, and responding to the data in a timely fashion, often in real time.

In this category of data we look for more complex behavior patterns in order to identify anomalous behavior. For example, we might look at a series of user actions such as entering one application, retrieving specific high-value data, then moving directly to another application, such as e-mail or a social network, particularly if there is a policy against personal use of the organization's computer and network resources during business hours. This might suggest that data are being exfiltrated (or leaked) by the user. The major challenge here is identifying what can be considered normal behavior, as described in [12].

Reasons for the Lack of Data

Beyond the costs of collecting, logging and analyzing security data (which can be significant), the basic reason that we do not have those data that are needed for tracing specific activities through applications is that there are few advocates among stakeholders. These stakeholders might include business owners, and software and security engineers. While software and security engineers are often very supportive of including logging and monitoring of security-related data, they do not usually have the requisite understanding of the applications and knowledge as to which data would be helpful in tracking activities. The business owners, on the other hand, do not understand what is needed for managing system security.

There is some question of our very ability to gather meaningful data and come up with relevant abnormal behavior patterns, which might suggest malicious or otherwise damaging activities. This perhaps suggests that we are not looking in the right

Table 2: Detection and Mitigation of Unauthorized Access and Activities

Category	Characteristics	Detection	Mitigation
Unauthorized access	Multiple failed attempts at logging on. Series of failed attempts followed by a successful logon.	Login to account of employees or customers who have left, but their accounts were not inactivated.	Create/enforce procedure to delete user accounts immediately when someone changes role or leaves. Check for anomalies such as more than one person trying to log on using the same credentials.
Inappropriately authorized access	High level of “trolling”, i.e., seemingly unrelated use of applications and access to data.	Atypical activities within applications and against databases. Numerous repeated attempts to access data for which the user is not authorized.	Restrict user access to applications’ functionality and data on a need-to-know basis Modify access rights as soon as role changes.
Authorized access, outside “normal parameters”	Seemingly excessive number of reads, writes, data exports for someone with a specific role.	Monitor use of critical applications and access to data. If a person leaves the organization, check activities retrospectively for unusual behavior.	Proactively – Be aware of, and respond to, significant changes in business environment, user responsibilities, etc. Responsively – Observe events and/or question users after events.
Authorized access, within “normal parameters”	No obvious deviation from normal activities.	Other aspects of behavior, such as dissatisfaction with job, company, recent changes, etc.	Introduce a process to get feedback about worker work attitudes and other influencing attributes.
Unauthorized access, outside “normal parameters”	Excessive number of logon attempts and/or attempts to get at unauthorized functions and data. Excessive number of reads, writes, data exports, and the like.	This should be detected before access is granted, but if not, then other behavioral discrepancies should be considered. Number of logon attempts, number and type of data accesses suggest anomalous behavior.	Mitigation includes combinations of processes to detect unauthorized access and anomalous behavior within systems and networks.
Unauthorized access, within “normal parameters”	No obvious deviation from normal activities.	Other aspects outside system, such as unusual times of activity.	Implement monitors that look for suspicious factors, such as time of day, day of week, frequency of access attempts, etc.

places. The most relevant information could well be external to the systems being monitored. For example, if someone quits an organization with little or no notice, or begins to behave erratically, or badmouths his or her organization and management, then it might behoove the organization to more carefully monitor that person's activities. Such activities include suddenly sending out streams of e-mails containing sensitive information. Such "symptoms" are more likely to suggest that certain information might be copied that otherwise would not.

Approach to Collecting the Data

Nevertheless, there may well be behaviors that do indicate improper activities, but data about them are just not collected. There is a need to define security data requirements and build the necessary data collectors into the applications during the design and development stages.

There has to be collaboration among software engineers, systems engineers, developers, application security experts, business owners, etc. But first we need to understand the various categories of security data: their characteristics, how they might be detected, and what must be done to reduce or eliminate the impact of the activities being recorded. We indicate these aspects in Table 2 for various categories of access.

While the Department of Defense exerts strong management in terms of who is authorized to access systems and maintains control over that access via the CAC, as mentioned above, there are still possibilities with respect to inappropriate registration of users and inappropriate use of those access credentials, as exemplified by the Manning case described above.

The Economics of the Proposed Approach

Costs of generating the data consist of the costs of the time and effort to include security data requirements, building the collectors in, populating the logs, analyzing the data, and so on. These can often be estimated fairly accurately.

On the other hand, it is very difficult, if not impossible, to determine specific and accurate returns on investment for building instrumentation into applications and developing metrics based on the new data generated and collected. Generally the justification for such efforts to improve security takes a few high-profile breaches, such as those mentioned above, where having had specific data might have prevented or mitigated an attack, or at least facilitated forensics efforts.

The costs and losses depend upon the entity, which was the victim, and its affiliations. However, there is a limit to the true out-of-pocket costs that an entity and its stakeholders (management, shareholders, employees, and customers) can incur. For example, the upper limit of loss, to which shareholders might be subjected, is the value of the stock they hold. Company officers and executives might be sued, which eventuality is often covered by insurance. However, management and other employees might lose their jobs resulting in monetary losses, psychological distress, etc. If the entity is forced to downsize or goes out of business, customers and business partners will have to find other sources and affiliations respectively, which might be difficult and expensive to achieve in certain markets.

In order to detect stealth behavior within applications, it is necessary to generate and analyze meaningful data about user activities. A major issue is that much of the required data is not created because the necessary data generators have not been incorporated into the application software.

Actual losses may far exceed what an entity and/or its stakeholders have the ability to pay. Thus a company might file for bankruptcy and any excess losses above what is covered by insurance and assets are borne by creditors or the public. This is in essence a case of "moral hazard" whereby the losses of the victim organization are limited. If another entity, such as the government, does not make up the difference between what the victim company can pay and what is believed to be the full cost of an incident, then part of the responsibility and uncovered costs fall upon individual victims and/or society at large. These are the so-called social or indirect costs.

Some Illustrative Examples

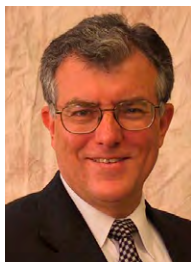
The author was involved in a situation where an institutional customer of a financial broker/dealer requested that the latter provide details of the activities within a specific system of a particular individual over a specified period of time. Not only was the broker/dealer unable to provide a report of generally available login and logout data because the data were comingled with the same information of other customers, but the application did not collect and log specific activities within the application itself. The author recommended that the requisite data collection capabilities be incorporated in a subsequent release of the application.

In another case, an organization detected hacking software residing on an employee's desktop computer. The employee was suspended pending the results of a forensic analysis, which took several weeks to complete. The result of the analysis was that the software had not been used. Here is a case where appropriate instrumentation might have avoided the unpleasantness of the employee's suspension altogether.

Summary and Conclusions

In order to detect stealth behavior within applications, it is necessary to generate and analyze meaningful data about user activities. A major issue is that much of the required data is not created because the necessary data generators have not been incorporated into the application software. In this article we have shown areas for improvement in data generation so as to increase situational awareness as it relates to stealth attacks against applications. While the process involves significant enhancements to the software engineering process, it is claimed that the resulting actionable information is well worth the effort. ♦

ABOUT THE AUTHOR



C. Warren Axelrod, Ph.D., is a senior consultant with Delta Risk, a consultancy specializing in cyber defense, resiliency and risk management. Previously, he was the chief privacy officer and business information security officer for U.S. Trust, the private wealth management division of Bank of America. He was a co-founder of the Financial Services Information Sharing and Analysis Center. Dr. Axelrod won the 2009 Michael Cangemi Best Book/Best Article Award for his article "Accounting for Value and Uncertainty in Security Metrics," published in the ISACA Journal, Volume 6, 2008. He was honored with the prestigious Information Security Executive Luminary Leadership Award in 2007. He received a Computerworld Premier 100 IT Leaders Award in 2003.

Dr. Axelrod has written three books, two on computer management, and numerous articles on information technology and information security topics. His third book is Outsourcing Information Security, published in 2004 by Artech House. His articles "Investing in Software Resiliency" and "The Need for Functional Security Testing" appeared in the September/October 2009 and March/April 2011 issues of CrossTalk magazine, respectively. He holds a Ph.D. in managerial economics from Cornell University, as well as an honors M.A. in economics and statistics and a first-class honors B.Sc. in electrical engineering, both from the University of Glasgow. He is certified as a Certified Information Systems Security Professional and Certified Information Security Manager.



Homeland Security

The Department of Homeland Security, Office of Cybersecurity and Communications, is seeking dynamic individuals to fill several positions in the areas of software assurance, information technology, network engineering, telecommunications, electrical engineering, program management and analysis, budget and finance, research and development, and public affairs. These positions are located in the Washington, DC metropolitan area.

To learn more about the DHS Office of Cybersecurity and Communications and to find out how to apply for a vacant position, please go to USAJOBS at www.usajobs.gov or visit us at www.DHS.GOV; follow the link Find Career Opportunities, and then select Cybersecurity under Featured Mission Areas.

REFERENCES

1. Axelrod, C. Warren, "Accounting for Value and Uncertainty in Security Metrics," ISACA Information Systems Control Journal (November 2008).
2. Scalet, Sarah D., "ChoicePoint Data Breach: The Plot Thickens – A timeline of key events surrounding the ChoicePoint data breach," CSO Online, May 2005. Available at <<http://www.csoonline.com/article/220341/choicepoint-data-breach-the-plot-thickens>>.
3. Vijayan, Jaikumar, "Update: Heartland breach shows why compliance is not enough – The huge data breach one year ago hammers home the need for multilayered security controls," Computerworld, January 2010. Available at <http://www.computerworld.com/s/article/9143158/Update_Heartland_breach_shows_why_compliance_is_not_enough>.
4. Jones, Penny, "NASDAQ brings in NSA to investigate breach – NSA involvement points to seriousness of 2010 hack, and importance of NASDAQ to US," DataCenterDynamics Focus, April 2011. Available at <<http://www.datacenterdynamics.com/focus/archive/2011/04/nasdaq-brings-in-nsa-to-investigate-breach>>.
5. Bradley, Tony, "Epsilon Data Breach: Expect a Surge in Spear Phishing Attacks," PCWorld, April 2011. Available at <http://www.pcworld.com/businesscenter/article/224192/epsilon_data_breach_expect_a_surge_in_spear_phishing_attacks.html>.
6. 2010 Data Breach Investigations Report, Verizon, Inc., 2010. Available at <http://www.verizonbusiness.com/resources/reports/rp_2010-data-breach-report_en_xg.pdf>.
7. Glithero, Bob, "From Insider Abuse to Insider Accountability: Identity Analytics Discover Insider Threats," Veriphys Inc., 2011. Available at <http://www.veriphys.com/download/Veriphys_WP_From-Insider-Abuse-to-Insider-Accountability.pdf>.
8. Homeland Security Presidential Directive 12/HSPD-12, 2004. Available at <http://www.cac.mil/assets/pdfs/HSPD_12.pdf>.
9. Galante, Joseph et al., "Sony Network Breach Shows Amazon Cloud's Appeal for Hackers," Bloomberg News. Available at <<http://www.bloomberg.com/news/print/2011-05-15/sony-attack-shows-amazon-s-cloud-service-lures-hackers-at-pennies-an-hour.html>>.
10. Overview available at <http://en.wikipedia.org/wiki/Bradley_Manning>.
11. Axelrod, C. Warren, "The Need for Functional Security Testing," Crosstalk, March/April 2011. Available at <<http://www.crosstalkonline.org/storage/issue-archives/2011/201103/201103-Axelrod.pdf>>.
12. NITRD (Networking and Information Technology Research and Development) Workshop, "Abnormal Behavior Detection Finds Malicious Actors," June 2011. Available at <http://www.nitrd.gov/fileupload/files/MaliciousBehavior_2011_NITRD_workshop.pdf>.

Developmental Automated Testing and Software Technical Risk Assessments

Brad Neal, SimVentions

Abstract. Testing continues to represent the single largest cost associated with the development of sophisticated, software intensive, military systems. A 20% reduction in overall testing cost, as requested by the Chief of Naval Operations (CNO) in his "Guidance for 2004," will save billions of dollars over the lifecycle of the myriad of tactical software systems. Such significant savings can only be achieved if the concept of testing begins very early in the development process.

Automated Testing

As tactical software development moves more toward open standards and becomes more focused on reducing development time and creating re-usable capabilities, the need for efficient and thorough testing of software becomes more critical than ever before. Testing artifacts can no longer be used only during initial development and then discarded. Coordinated incremental software builds and highly re-usable software testing capabilities will allow developers to coordinate and reuse software tests and testing tools repeatedly. Automated software testing leverages tools to stimulate the software under test, measure its response, and assess the correctness of its response without significant human intervention.

Automated Testing (AT) tools and systems will be based on affordable COTS hardware and software tools that allow more flexible and repeatable use. These capabilities are directly enabled, and necessitated by the proliferation of Open Architecture software and computing environments and standards. This is in contrast to traditional testing methodologies and systems that require specialized hardware to test new software systems and component interfaces. AT will investigate and enable automated, affordable, rapid, high quality, and reusable testing of large software-based systems throughout the entire lifecycle.

Determining the benefit of an AT program requires the identification of the business case for investing in automated test. Simply stated, the business case for AT stems from The CNO Guidance for 2004 [1], which included direction to the Commander, Operational Test and Evaluation Force (COMOPTEVFOR) to lead a collaborative effort among Navy, OSD, and contractors to reduce the costs of Test & Evaluation (T&E) by 20%. In developing a response to the CNO Guidance for 2004, COMOPTEVFOR surveyed programs and included the following as T&E cost drivers:

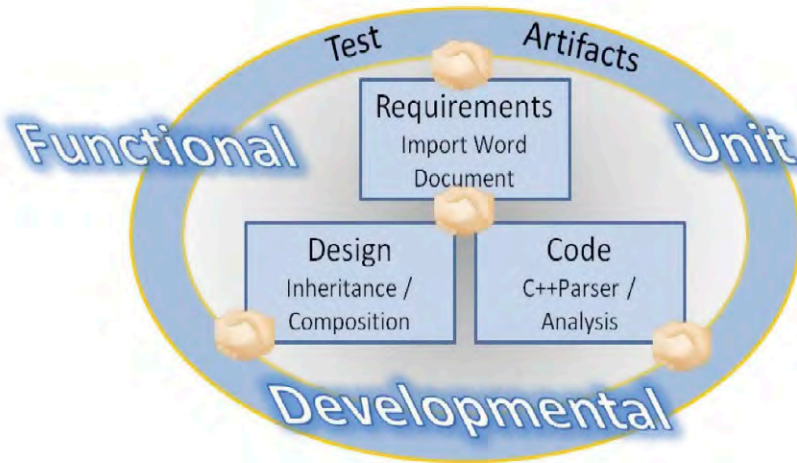
- Redundant testing
- Increased levels of regression testing driven by technology insertion
- Increasing complexity of computer software testing
- Interoperability testing and certification
- Development of unique /duplicate facilities and test beds

This helps to identify the need, the measures necessary to detect progress, the impact to the program under test and the potential ROI for the program. The technical process identifies the steps that are followed that will prove or disprove the business case. The enabling technology involves identifying the tools, targets, and standards for carrying out the technical process. Finally, implementation involves physical execution of the enabling technology, a final determination of whether or not AT is feasible, beneficial, and validates the anticipated ROI, and where AT could be implemented.

Yet, currently there are no policies governing the use of AT tools and techniques within the Acquisition Framework; the DoD 5000 series instructions do not include any reference to AT. However, with the emergence of common product lines, a relationship between the acquisition community and AT concepts needs to be firmly established before a marriage into contractual language is produced. The Navy is going into uncharted territories and studies are needed, pilots performed, and proofs of concepts explored to discover applicability and affordability.

In an effort to take an expanded approach to AT, leveraging the knowledge and insight gained from the Object Management Group (OMG) standards work, an attempt at injecting automation in other areas not greatly explored within industry or Navy programs was attempted, i.e.: 1) Requirements, 2) Design, 3) Code, and 4) Test Coverage. The ability to automatically generate the Requirements Traceability Matrix from top-level requirements and intermediate designs to the actual Computer Program (CP) would eliminate the need to perform this task manually as is often done within disparate programs; DOORS, Excel, Enterprise Architect, and Integrated Development Environments. Standards organizations like the OMG will be leveraged to engage industry, academia, and government communities by developing options and approaches via Request for Information, Request for Proposal (RFP), technology pilots, and white papers. The right mix of companies, organizations, and Navy industry relationships will enable an open and streamlined process for technical standards establishment. If standards are adopted, usability is likely to increase bolstering the business case for AT.

Figure 1: Information Exchange Handshake



Software Assessments

An initial implementation, performed in FY '09, involved the demonstration of a graphical view of requirements mapped to source code. This effort resulted in the generation of a Developmental Automated Risk Testing (DART) tool, which automates the process of mapping top-level requirements to lower level requirements, software design and CP code allowing the user to traverse bi-directionally to examine all the software artifacts associated with each requirement. DART offers automated information exchange handshakes (Figure 1) coupling functional, unit, and operational testing with developmental information. If the requirements change, this tool will allow the developer to instantly see what portion of the CP is affected along with test data specifics. It will allow a tester / analyst / developer to trace an error encountered in a component under test to its specifications and verify its function; determine if the error is in the CP or in the specification; and if so, whether the parent requirements for these specifications also need to be evaluated.

Built-in automated metrics such as code complexity, afferent/efferent instability, along with code fluidity evaluations aid the developer in identifying potential problem areas early. An overall risk assessment offers probability and impact measures based on issues with the code. This analysis takes into account complexity, orphans, and density of complex code among other things. A technical risk chart provides a quick look into the software for stakeholders, providing a status that updates as bugs are fixed and the design is analyzed.

Linking requirements to code provides continuity between the beginning and the end of development and offers the ability to automate the testing of your software design. It offers high-level automated software debugging and promotes standards in requirements documentation and software development. Below are some example use cases subject to analysis for a sample program under test:

- DART has a high probability of making code/design correct by introducing a Level 0 "smoke test" which focuses on coverage rather than on functionality.
- DART facilitates the developers job of having to open up a requirements document, design, or code; all are shown in a dashboard with the automated metrics decreasing time spent using disparate support programs to view the data.

- DART's automated complexity metric determines if code is testable; high complexity potentially introduces risk into program testing and threatens quality.
- DART's automated code fluidity metric is a functional software flow check determining if pieces of code are either called or are calling another piece of code; if not, the developer needs to verify necessity or completeness.
- DART can be used during peer reviews to "grade" developers (determine how the overall code structure and design looks to determine if the programmer has successfully created a testable design).
- DART offers insight into a programmer's design mechanics indicating their ability to meet program software compliance while assessing their coding competence.
- DART offers an immediate quick look mechanism for code and shows how a program satisfied a requirement technically allowing for future software reuse where capability is needed enterprise wide.

Identifying the association between the design, development, and transition of a capability in a common dashboard offers an opportunity to perform automated metric and report generation, analysis of requirements coverage, and testing as a whole. This will enable program personnel to gain firsthand experience in discovering the "need to fleet" associations, unleashing the potential of automated testing. Within the standard Systems Engineering V-chart for the development of capabilities, systems, and platforms, there should be more of a relational exchange of information, not a waterfall of data dumped from one activity to the next. Feedback loops from the transition or testing side of the V-chart should be tied directly to requirements, system design, and CP code for true testing accountability. This traceability into the systems engineering process allows for visibility into the artifacts associated with each piece of development. Only when this traceability is available can AT policies and procedures be introduced and employed to demonstrate true acquisition efficiency.

Contract Language

The majority of the contracts start at milestone B, where concept exploration is complete and technology is determined to be mature enough. Therefore, the stakeholders accountable for all T&E activity and responsible for creating the Test and Evaluation Strategy (TES), which is the initial planning document describing test activities throughout the acquisition lifecycle, are prepared to make key decisions on what is best for streamlining their test process from cradle to grave. Articulating in the TES that AT capability should be used could prove to be a major influence on contract language. This TES document is a key item to be considered when developing the contract Statement of Work for the RFP. The Office of the Secretary of Defense (OSD) T&E Contract Guide states:

"Acquisition planning is the process of identifying and describing contract requirements and determining the best method for meeting those requirements including solicitations and contracting. During the program lifecycle it is critical that the Project Manager (PM), systems engineer, and T&E personnel recognize that early and consistent incorporation of T&E considerations

and requirements begins at the onset of program planning during the Material Solutions Analysis and Technology Development phases [2].

Raising awareness and educating the T&E stakeholders involved with creating the TES through a conclusive business case analysis and identifying what the ROI is for AT will begin to influence policy consequently effecting contract requirements. Specific metric weighting based on stakeholders, as shown in the acquisition process below (Figure 2) (PM, Warfighter, Contractor, and T&E Tester), are provided throughout the rest of the document.

Metrics

The word metric has many different meanings; the following definitions provide a basis for this discussion. A metric is, "A system of related measures that facilitates the quantification of some particular characteristic" [3]. Implicit in the use of the word metric is the notion of a standard means of measuring something in a known space. Further calibration of the term metric comes from the definition, "The application of statistics and mathematical analysis to a field of study" [4]. This definition indicates the need for sufficient rigor and sensitivity in the description of the measures to provide meaningful quantitative results. These results can then be used to support various analyses such as ROI. Within each metric, a description and explanation of how it relates to the AT business case is provided along with any relevant categories in the context of DART.

Metric 1: Time

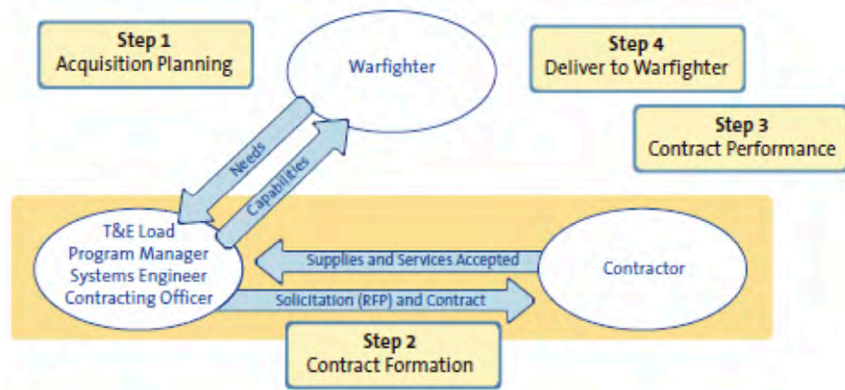
Time is a simple measure. It is well defined, understood, and standardized. While most of the systems being tested using AT technology are concerned with very small units of time such as milliseconds, the AT process must be focused on a much larger scale. In the context of AT, time has most to do with the calendar period need to perform a certain activity. As a business case metric, the timeframes of meaning typically span days, weeks, months, or quarters.

There are two primary categories of time, labor time and calendar time. Both factors are important to keep in perspective. Labor time is directly related to cost. Calendar time is directly related to delivery completion and risk. A reduction in either of these categories can be seen as a clear improvement in the ROI equation. Depending on the stakeholder, time impact varies. Time to the developer relates to earned value, contract management, and continuation. Time to the PM relates to meeting planned delivery to the platform manager and affects his measure of performance. Time to the platform manager relates to meeting planned ship delivery schedules, which can relate to significant penalty cost by a shipbuilder and/or lack of needed war fighting capability to the fleet. Time to the operational tester relates to certifying the operational effectiveness and safety associated with fleet use.

Metric 2: Delivery Risk

Delivery Risk is a measure of how likely a particular issue will negatively affect the ability to complete a project. This metric provides a qualitative adjustment to other quantitative metrics outlined in this document. Therefore, any assessment of delivery risk will be linked to an associated sub-category to which it belongs: Time/schedule impact, cost of system, and complexity/

Figure 2: Acquisition Process



difficulty of implementation. There are several reasons that an issue can have an impact on delivery risk:

- * If the number of functional requirements is very high
- * The design complexity causes difficulty in procurement or development
- * The number of defects detected is too high
- * The schedule is aggressive
- * There is insufficient funding

Time, schedule, and cost are well-understood factors, and are discussed in this document. There is a value in capturing the qualitative impact of these related to AT as well as the quantitative means described above. Difficulty of implementation is a category of risk that measures any extra effort, special skills, processes, equipment, or other factors that complicates (or simplifies) the execution of the project. In the context of AT, the question one must ask is, "How much delivery risk (increased or decreased) is introduced into my system as a result of AT?" This question should be asked in the context of time, schedule, and cost. Given the system under test and the stakeholder's viewpoint, the answer may have either a positive or negative affect on the ROI equation.

Metric 3: Quality

Quality is defined as the level by which a developed item meets or exceeds the requirements, endurance, and reliability specified in the design. The overall prevalence of quality software can be directly correlated to the magnitude of what has already been discovered, and the extent to which the software meets mission objectives can be measured. There are several characteristics of quality that can be considered:

- Opening and closing rate of defect reports
- Prevalence of defects
- Predictable/consistent performance
- Operational (reliable, available)
- Functional requirements coverage

Metric 4: Cost

Cost is the amount of expenditure required to complete an activity. There are many task contributors that impact the amount of cost incurred. This section divides the types of cost typically incurred and tracked in activities related to test efforts. These categories are not mutually exclusive, nor exhaustive of the cost elements that are possible.

There are many ways to stack and compare cost, or colors of money:

- Non-recurring versus recurring
- Operations and maintenance versus development
- Research and development maturity level (6.1, 6.2, 6.3, 6.4, etc.)
- Labor versus other direct costs

Various stakeholders' views provide different perspectives on which cost trade off is relevant, and what they feel is most important. Below are some categories of cost identified: Development cost, labor cost, license/equipment cost, and facility cost. Cost is simply measured in dollars. The scale of cost-relevance is proportional to the scale of the project. Cost relative to time saved or time lost will be shown in accessing each separate metric in parallel.

ROI

All relevant metrics need to be considered when the end goal is to deliver a better quality capability to the fleet faster, cheaper and at lower risk. In order to establish this level of ROI, a relationship between the different metrics must be established to determine a relative weighting of one metric to the next. This is a difficult task since different factors will weigh the importance of the metrics differently. The relationship is, therefore, stakeholder and T&E step dependent. An initial weighting may be established by polling the stakeholders, yielding a composite weighting, and using that through the life of the program. An alternative would be to have each stakeholder group perform their relative weighting of metrics and compute an ROI for each step based on the expectation from preceding programs.

An additional consideration must be a predicted effect on metrics for each stakeholder's responsibility and how this affects their desire to implement AT. As an example, DART would require developers to implement traceability in support of development and testing, therefore delivery may incur an additional cost to the program. This negative effect on cost and potentially time spent must be offset by a combination of positive effects on the other metrics of interest to them. If a relationship can be shown that utilizing DART will effectively reduce completion risk and increase quality sufficiently, then it becomes of interest to the PM. If it does not have this effect, it will be an uphill struggle to get PMs to accept the requirement.

Another benefit to automating testing is a higher confidence in the coverage of requirements during test. Many current and traditional testing methodologies, despite considerable effort, yield a wide degree of variability in the function points actually tested. It is believed that by utilizing automated tests (i.e. DART), one will know what has been tested, yielding higher confidence that critical items have been checked.

Summary

Based on the COMOPTEVFOR findings and the imminent change in paradigm to open architecture, there is a need for a more flexible and innovative way to improve synergy throughout the development and T&E communities within the acquisition framework. Traditional systems engineering process and contractual language evolution is needed; more effective ap-

proaches for testing are required to meet the CNO guidance for reducing T&E costs by 20%. However, as the business case for AT is evaluated more closely, it is apparent that there are metrics beyond cost that need to be considered when looking at the overall ROI associated with the desire to deliver tactical war fighting capability faster and cheaper while at the same time providing a higher quality product with equivalent functionality.

As the AT community grows and is leveraged throughout DoD, the business case for utilizing tools and techniques such as DART will need to be assessed along with its ease of use. Technical and enterprise maturity will need to be cataloged to establish a baseline for progress throughout component and system development. A program willing to identify and support development of standards for AT and bring key stakeholder metrics to the forefront will establish a true overall benefit and identify the impacts (both positive and negative) that AT has on its development. ROI metrics will help discover and establish the technical process and contract language for the application of AT to major software development efforts. ♦

ABOUT THE AUTHOR



Mr. Brad Neal has 13 years of systems engineering experience supporting various DoD program offices and is currently the engineering services group lead at SimVentions. Efforts involve developing technologies to facilitate rapid technology transition and conducting business case analysis on current and traditional T&E processes to identify areas of opportunity for automation to maximize ROI. Brad has performed many automated software assessments discovering technical risk within a programs acquisition lifecycle.

REFERENCES

1. Department of Defense. CNO Guidance for 2004. 2004. February 2010 <http://www.globalsecurity.org/military/library/policy/navy/cno-guidance_2004.htm>.
2. Department of Defense Directive 5000.1. Directive. Washington, D.C.: Department of Defense, May 12, 2003.
3. Dictionary.com. Metric. 2010. February 2010 <<http://dictionary.reference.com/browse/Metric>>.
4. Metrics. 2010. February 2010 <<http://dictionary.reference.com/browse/Metrics>>.

Cyber Strategy, Analytics, and Tradeoffs: A Cyber Tactics Study

Don O'Neill, Independent Consultant

Abstract. A framework of cyber tactics is suggested in terms of intended function and input and output semantics for each tactic including anticipation, detection, attribution, and counter measures. Cause and effect chain elements spanning goals, weaknesses, attributes of building security in, attack outcomes, bad actors, and consequences are identified and traced for selected cyber tactics. Consequences are traced for selected goals and attack outcomes.

Cyber Strategy

There is a need for plain talk on cyber security. When it comes to cyber attacks, there is no safe harbor. 75% of the time organizations learn that they have been the targets of an attack from others outside their organization¹. So much for awareness and readiness!

In "Meeting the Cyber Security Challenge" [1], the authors assert that challenges in culture, governance, shared ownership, and accountability are larger than the challenges of IT. This brings to mind some unanswered questions:

1. These broader challenges notwithstanding, how close are we to actually solving the challenges of IT with respect to cyber security?
2. How can we change the game so that bad actors are not on the same sure footing as legitimate users?
3. Can the answer be found in engineering innovation or do we have to resort to legal frameworks?
4. If national competitiveness, security, and privacy are competing goals, what are the priorities?

Some guiding definitions are in order to cope with item 4.

1. Competitiveness is the ability of a corporation to meet the test of international markets while maintaining or boosting the wages of its workers.²
2. Security is the condition of being protected against danger or loss.
3. Privacy is the ability to reveal oneself selectively.

This is the stuff of cyber strategy. We need a cyber strategy to anticipate, avoid, withstand, mitigate, and recover from the effects of adversity whether manmade or natural under all circumstances of use. In the current environment, there is little focus on anticipation and avoidance before the attack, only the need for first responders after the attack. These first responders arrive in the form of dedicated software engineers who must attend to cleanup and recovery chores...and more patching, distraught business executives who must make up for lost opportunity costs and try to overcome the impact of loss of trust in

their operations; more frustrated users who must cope with loss of availability through jerrybuilt workarounds; and more victimized customers who must suffer loss of privacy.

Cyber attacks represent a clear and present danger and a threat to the competitiveness and security of the nation. Yet our cyber strategy has been primarily driven by politics. Just consider the stresses surrounding privacy and security where privacy is the freedom and ability to reveal oneself selectively and security is the condition of being protected against danger or loss, both reasonable goals but on a collision course nevertheless.

Analytics

Professor Davenport refers to analytics as, "The extensive use of data, statistical and quantitative analysis, exploratory and predictive models, and fact-based management to drive decisions and actions" [2]. Data is exploding in every field. The knowledge of the world is reflected in data that can be processed by mathematical models. This process of analytics can help to understand the past and attempt to anticipate the future³.

Exploiting the technology of fact-based decision-making through a large-scale analytics program in the cyber security domain would be a powerful way to engage executives in the business case of cyber security with particular attention on the nation's critical infrastructure and the defense industrial base.

The cyber tactics study is designed to trace useful cause and effect chains connecting the tactics of anticipation, detection, attribution, and counter measures across several dimensions including organization goals, standard of excellence of software engineering practice, common weaknesses and vulnerabilities, cyber attack outcomes, bad actors, and consequences.

While the semantics of these dimensions have been specified as a first step towards framing the data collection requirements for an analytics project, the plan is to stimulate use of Monte Carlo simulation to assess the probability of outcomes and consequences and use of multivariate analysis on combinations of independent variables to understand the effect on a single variable.

Tradeoffs

A strategy is an overarching plan to achieve a vision along with the policies and protocols that link and coordinate the tactics employed. In cyber security time matters and things move fast, at the speed of light...faster than people can close their command and control loops. So a cyber strategy must include the policy to authorize in advance time critical actions based on tradeoffs that have carefully weighed cause, effect, and consequences. It is through continuously rebalanced tradeoffs and adjustments in the policy authorizing time critical actions that a strategy retains its currency in the face of new knowledge and new threats.

A cyber strategy is a policy composed of a collection of defined tradeoffs made in advance with implementing tactics ready for deployment. For example:

1. Where the inevitability of a cyber attack is a given, the power of trading off acceptance of some consequences in order to avoid or limit other consequences should not be ignored. How much lost opportunity, loss of availability, and loss of privacy are we willing to trade off to avoid cleanup and recovery costs and loss of trust impact?

2. The power of the shutdown as a tactical instrument to trade off consequences should not be underestimated. Upon shutdown, we immediately incur lost opportunity and loss of availability in order to avoid or limit cleanup and recovery costs and loss of trust and loss of privacy impacts.

3. The power of the social contract as an instrument to implement trade offs should not be overlooked. Where civility is defined as the sacrifices we make for others, perhaps an appeal to civility would prompt people to agree to sacrifice some level of privacy in order to promote a higher level of security for others. Here civility would lubricate the trade off between security and privacy. However, the catalyst needed to spark the social contract is leadership, which is not always available.

Trading off consequences is situational. For example, consider the priority ranking of consequences by reputation, economics, and mission in Table 1. There are consequences in prioritizing consequences.

1. In the reputation scenario, the highest consequences to avoid are loss of trust and loss of privacy followed by lost opportunity and loss of availability and then cleanup and recovery. The financial services sector where trust is all important fits the reputation scenario.

2. In the economics scenario, the highest consequences to avoid are lost opportunity and loss of availability followed by cleanup and recovery and then loss of trust and loss of privacy. The energy sector fits the economics scenario.

3. In the mission scenario, the highest consequences to avoid are lost opportunity, loss of availability, and loss of trust followed by loss of privacy and then cleanup and recovery. The telecommunications sector fits the mission scenario.

Cyber Tactics Framework

If cyberspace is to be a battleground, we need to stop playing possum and formulate a game changing cyber strategy, one that is informed and buttressed by cyber tactics. Dennis C. Blair, Director of National Intelligence, said, "Intelligence agencies are integrating cyber security with counterintelligence and bolstering the ability to understand, detect, attribute and counter cyber threats" [3]. In addition, intelligence agencies also need to anticipate!

These cyber tactics (Figure 1) span anticipation [4], detection, attribution, and counter measures (Table 2). However, these cyber tactics are currently underdeveloped and insufficient as implementers of the nation's cyber strategy. Until these cyber tactics evolve to a more robust level of professional maturity, the nation's cyber strategy will continue to be framed in the political domain with its unresolved clashes of philosophy. Without a framework of cyber tactics, the nation's cyber strategy will continue to remain unhinged.

Anticipation

1. The intended function of anticipation is to make decisions about the future based on expectation in order to anticipate, avoid, withstand, mitigate, and recover from the affects of adversity under all circumstances. Unlike other tactics, anticipation takes place before an attack.

2. Cause and effect forward chain traces are used in order to interdict cause primarily in terms of cyber security goal selection, standard of excellence attributes for software engineered products, and economic cost impact and end user mission loss consequences. A secondary cause and effect thread composed of common weaknesses, attack outcomes, and bad actors may also inform anticipation tactics and serve to verify the primary thread.

3. The pursuit of anticipation tactics contributes to an understanding of what and how in terms of weaknesses and vulnerabilities, build security in maturity, and security in depth.

Detection

1. The intended function of detection spans digital situation awareness, operation sensing and monitoring, and the identification of defects, weaknesses, vulnerabilities, attacks, outcomes, and bad actors through monitoring, inspection, assessment, deep analytics, surveillance, alertness, and awareness.

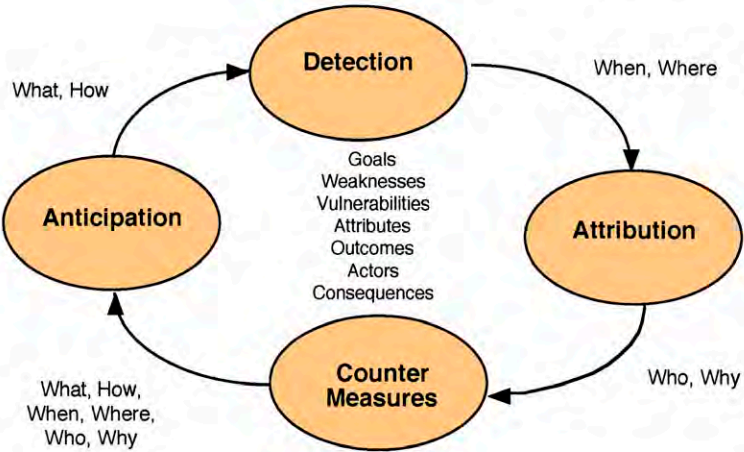
2. The collection of dynamic operations sensing and monitoring logs fuels an activity of deep analytics. The systematic inspection of lifecycle systems and software artifacts by experts reveals defects, weaknesses, and vulnerabilities from which attacks are composed [5]. The automatic static analysis of software code, both source and object, also fuels an activity of deep analytics [6]. As common weaknesses, common vulnerabilities, and common attack profiles emerge, they become patterns and templates for subsequent surveillance.

3. Detection tactics contribute to an understanding of when and where in terms of defects, attacks, and outcomes.

Table 1: Priority Ranking of Consequences

Consequence/Priority	Reputation	Economics	Mission
Cleanup	3	2	3
Lost Opportunity	2	1	1
Recovery	3	2	3
Loss of Availability	2	1	1
Loss of Trust	1	3	1
Loss of Privacy	1	3	2

Figure 1: Trusted Pipe™ Cyber Tactics Framework



Attribution

1. The intended function of attribution is focused on the assessment of cause and effect trace artifacts to identify a person, account, group, or intermediary responsible or involved [7].
2. Cause and effect backward chain traces are used in order to profile the cause in terms of attack outcomes, bad actors [8], and economic cost impact and end user mission loss consequences.
3. Attribution tactics contribute to an understanding of whom and why in terms of motivation and rationale, exploits employed, intended attack outcomes, and consequences sought.

Counter Measures

1. The intended function of counter measures is focused on the detection and elimination of common weaknesses; adherence to rigorous standard of excellence attributes of completeness, correctness, and rules of construction; elimination of attack outcomes; strengthening defenses against bad actors; and avoiding, withstanding, mitigating, and recovering from consequences.
2. Actual when and where in terms of defects, attacks, outcomes, and actual who and why in terms of motivation, intended attack outcomes, and consequences sought provide the fuel for composing effective counter measures.
3. Counter measures tactics are focused on understanding

what and how, when and where, and who and why in terms of counter measures to interdict weaknesses and vulnerabilities, to build security in, and install security in depth.

Cause and Effect

The primary cause and effect chain elements in play for each cyber tactic are identified in Table 3.

1. Anticipation focuses on a thread of goal commitment, standard of excellence attributes, cost element and end user mission loss consequences. As an organization commits to more demanding goals leading to more strict adherence to the standard of excellence attributes of completeness, correctness, and rules of construction, the likelihood of cost element and end user mission loss consequences diminish.
2. Detection focuses on a thread of common weaknesses, attack outcomes, and bad actors. As an organization obtains the deepest possible understanding of common weaknesses and how these are used by bad actors and contribute to attack outcomes, the organization capability to detect and defend itself is advanced.
3. Attribution focuses on a thread of attack outcome, bad actor, cost element and end user mission loss consequences. As an organization is better able to associate attack outcomes

Table 2: Trusted Pipe Cyber Tactics Semantics

Cyber Tactics	Intended Function	Input	Output
Anticipation	Making decisions about the future based on expectation in order to anticipate, avoid, withstand, mitigate, and recover from the effects of adversity under all circumstances	Cause and effect forward chain trace in order to interdict cause primarily in terms of Cyber Security goals, standard of excellence attributes, and economic cost impact and end user mission loss consequences and secondarily in terms of common weaknesses, attack outcomes, and bad actors	Understanding what and how in terms of weaknesses and vulnerabilities, build security in maturity, and security in depth
Detection	Digital situation awareness, operation sensing and monitoring, and identification of defects, weaknesses, vulnerabilities, attacks, outcomes, and bad actors through inspection, assessment, deep analytics, surveillance, alertness, and awareness	Dynamic operations sensing and monitoring logs for deep analytics; life cycle systems and software artifacts for expert inspection of defects, weaknesses, and vulnerabilities; software code both source and object for automated static analysis and deep analytics; common weaknesses, common vulnerabilities, and common attack profiles for assessment	Understanding when and where in terms of defects, attacks, outcomes
Attribution	Assessment of cause and effect trace artifacts to identify a person, account, group, or intermediary responsible or involved	Cause and effect backward chain trace in order to profile causes in terms of attack outcomes, bad actors, and economic cost impact and end user mission loss consequences	Understanding who and why in terms of motivation and rationale, exploits employed, intended attack outcomes, and consequences sought
Counter Measures	Detect and eliminate common weaknesses; adhere to standard of excellence attributes; eliminate attack outcomes; strengthen defense against bad actors; avoid, withstand, mitigate, and recover from consequences	Actual when and where in terms of defects, attacks, outcomes; actual who and why in terms of motivation, intended attack outcomes, and consequences sought	Understanding what and how, when and where, and who and why in terms of counter measures to interdict weaknesses and vulnerabilities, to build security in, and install security in depth

Table 3: Identifying Cause and Effect Elements by Cyber Tactics

Cause and Effect Chain	Elements	Anticipation <i>What, How</i>	Detection <i>When, Where</i>	Attribution <i>Who, Why</i>	Counter Measures <i>What, How, When, Where, Who, Why</i>
Goals	1. Anticipating 2. Avoiding 3. Withstanding 4. Mitigating 5. Recovering	●			
Weaknesses	1. Insecure Interaction Between Components 2. Resource Management 3. Porous Defense		●		●
Attributes	1. Completeness 2. Correctness 3. Rules of Construction	●			
Outcomes	1. Unauthorized Access 2. Loss of Data 3. Tampering with Data 4. Erosion of Performance 5. Denial of Service		●	●	●
Actors	1. Inadvertent Actor 2. Disgruntled Employee 3. Hacker 4. Corporate Spy 5. Criminal 6. Terrorist 7. Organized Crime 8. Nation State		●	●	●
Consequences	1. Cleanup 2. Lost Opportunity 3. Recovery 4. Loss of Trust 5. Loss of Availability 6. Loss of Privacy	●		●	●

and bad actors, the organization will be better equipped to determine who launched the attack and the motivation and rationale behind it both useful in assessing the broader implications of the attack, the likelihood of whether future attacks should be expected, and the need for law enforcement and national security authorities to apprehend the bad actor.

4. Counter measures focus on a thread of common weaknesses, attack outcomes, bad actors, cost element, and end user mission loss consequences. As an organization is better able to coordinate its anticipation tactics with detection and attribution tactics in answering the questions of what, how, when, where, who, and why, it is able to introduce counter measures that effectively reduce the consequences of a cyber attack.

Consequences

Cause and effect chain consequences are traced for selected cyber tactics in Table 4a and 4b.

Considering the cyber tactic of anticipation, the goals to which an organization makes a commitment determine the consequences that result (Table 4a).

1. Starting at the top of the cause and effect chain with a commitment to the goals of anticipating and avoiding, the standard of excellence attributes for completeness, correctness,

and rules of construction are expected to be met, and the consequences associated with cost element impacts and end user mission impacts are more likely to be avoided.

2. With a commitment to the goal of withstanding, the standard of excellence attributes for completeness, correctness, and rules of construction are expected to be met, and the consequences associated with lost opportunity cost and end user mission impacts are more likely to be avoided.

3. With a commitment to the goals of mitigating and recovering, the standard of excellence attributes are unlikely to be met, and the consequences associated with cost element impacts and end user mission are not likely to be avoided.

Considering the cyber tactic of attribution, the attack outcome provides a likely indication of the type of bad actor and the consequences (Table 4b).

1. Moving down the cause and effect chain to attack outcomes, unauthorized access may be attributed to all bad actors except the inadvertent actor with the expected consequences of loss of trust and loss of privacy impacts.

2. Loss of data may be attributed to all bad actors with the expected consequences of lost opportunity and recovery costs, and loss of trust and loss of privacy impacts.

3. Tampering with data may be attributed to an inadvertent

Table 4a: Tracing Cause and Effect Consequences by Anticipation Tactic

Cause and Effect Chain	Elements	Goal	Goal	Goal
Goals	1. Anticipating 2. Avoiding 3. Withstanding 4. Mitigating 5. Recovering	Yes Yes	Yes	Yes Yes
Attributes	1. Completeness 2. Correctness 3. Rules of Construction	Yes Yes Yes	Yes Yes Yes	
Consequences	1. Cleanup 2. Lost Opportunity 3. Recovery 4. Loss of Trust 5. Loss of Availability 6. Loss of Privacy		Yes Yes	Yes Yes Yes Yes Yes Yes

Table 4b: Tracing Cause and Effect Consequences by Attribution Tactic

Cause and Effect Chain	Elements	Attack Outcome	Attack Outcome	Attack Outcome	Attack Outcome	Attack Outcome
Outcomes	1. Unauthorized Access 2. Loss of Data 3. Tampering with Data 4. Erosion of Performance 5. Denial of Service	Yes	Yes	Yes	Yes	Yes
Bad Actors	1. Inadvertent Actor 2. Disgruntled Employee 3. Hacker 4. Corporate Spy 5. Criminal 6. Terrorist 7. Organized Crime 8. Nation State	Yes Yes Yes Yes Yes Yes Yes Yes	Yes Yes Yes Yes Yes Yes Yes Yes	Yes Yes Yes Yes Yes Yes Yes Yes	Yes Yes Yes Yes Yes Yes Yes Yes	Yes Yes Yes Yes Yes Yes Yes Yes
Consequences	1. Cleanup 2. Lost Opportunity 3. Recovery 4. Loss of Trust 5. Loss of Availability 6. Loss of Privacy	Yes Yes Yes	Yes Yes Yes Yes Yes	Yes Yes Yes Yes Yes	Yes Yes Yes Yes Yes	Yes Yes Yes

actor, disgruntled employee, hacker, and terrorist with the expected consequences of cleanup, lost opportunity, recovery costs, and loss of trust impacts.

4. Erosion of performance may be attributed to a disgruntled employee, hacker, and terrorist with the expected consequences of cleanup, lost opportunity, recovery costs, and loss of trust and loss of availability impacts.

5. Denial of service may be attributed to a disgruntled employee, hacker, terrorist, and nation state with the expected consequences of lost opportunity and loss of availability impacts.

Mapping Policy to Implementation

The cyber tactics of anticipation, detection, attribution, and counter measures provide a strategic policy framework of stretch goals. The adoption of these tactics by an organization represents strategic intent since the implementation of these tactics is currently underdeveloped and needs to evolve to a higher level of professional maturity. Nevertheless, some discussion is called for to begin to bridge the gap between policy and practical application.

This discussion of practical application is centered around selected indicators of good software engineering, known common weaknesses and vulnerabilities, and observed bad actor behavior:

1. Assuring trustworthiness by building security in during construction and using standard of excellence indicators for completeness, correctness, and rules of construction as a guide.

2. Assuring security by protecting from known common weaknesses and vulnerabilities using the Top 25 Most Dangerous Programming Errors report [9] as a guide.

3. Assuring protection by understanding the behavior of bad actors using the FBI identification of cyber criminal behavior as a guide.

Standard of Excellence

The standard of excellence is a set of indicators intended to distinguish good systems and software engineering practice associated with software product engineering. The indicators were first introduced in association with the system of checklists used to assist in the conduct of the Software Inspections Lab [10]. More recently the full system of evolved checklists appears in

Common Weakness/Attack Outcome	Unauthorized Access	Loss of Data	Tampering with Data	Erosion of Performance	Denial of Service
Insecure Interaction Between Components	4	7	3	2	1
Resource Management	2	8	9	9	3
Porous Defenses	6	5	6	0	1
Total yes	12	20	18	11	5

Table 5: 2010 Common Weakness Group/Attack Outcome

a book chapter entitled “Inspections an Upfront Quality Technique” on pages 156-161 [11]. In addition a selected subset of the standard of excellence checklists appeared in the article on “Peer Reviews” in the Encyclopedia of Software Engineering [5].

Tracing the essential factors of cyber security begins with the proposition that security must be built in and that trustworthiness is achieved when certain standard of excellence attributes are assured.

1. Completeness is based on traceability among software product artifacts of various types including requirements, architecture, specifications, designs, code, and test procedures. Completeness analysis may be assisted by tools that trace the components of a product artifact of one type to the components of another type. Completeness analysis of predecessor and successor artifacts reveals what sections are missing and what fragments may be extra. A byproduct of the completeness analysis is a clear view of the relationship of requirements to the code product: straightforward (one to one), simple analysis (many to one), and complex (one to many).

- a.** Has traceability been assessed?
- b.** Have all predecessor requirements been accounted for?
- c.** Were any product fragments revealed not to have traceability to the predecessor requirements?
- d.** Was traceability found to be straightforward, simple, or complex?

2. Correctness is based on reasoning about programs through the use of informal verification and correctness questions derived from the prime constructs of structured programming and their composite use in proper programs. Input domain and output range are analyzed for all legal values and all possible values. State data is similarly analyzed. Adherence to project specified disciplined data structures is analyzed. Asynchronous processes and their interaction and communication are analyzed.

- a.** Is the function commentary satisfied?
- b.** Are programs limited to single entry and single exit?
- c.** Is the loop initialized and terminated properly?
- d.** Does the input domain span all legal values?
- e.** Is there systematic exception handling for illegal values?
- f.** Are disciplined data structures used?

3. Rules of construction are based on the software application architecture and the specific protocols, templates, and conventions used to carry it out. For example, these include interprocess communication protocols, tasking and concurrent

operations, program unit construction, and data representation.

- a.** Are guidelines for program unit construction followed?
- b.** Is the interprocess communication protocol followed?
- c.** Are data representation conventions followed?
- d.** Is the system standard time defined and followed?
- e.** Are encapsulation, localization, and layering used to achieve object orientation?
- f.** Is logical independence achieved through event driven and process driven paradigms, late binding, and implicit binding?
- g.** Is scalability achieved through uniformity, parameterization, and portability?
- h.** Are fault tolerance, high availability, and security achieved?

Common Weaknesses

Managed by the MITRE Corporation and the SANS Institute, a collection of industry experts supported by the NSA and Department of Homeland Security's National Cyber Security Division convened to agree on the 25 Most Dangerous Programming Errors [9]. The degree to which trustworthiness is achieved is indicated by the defects that make up the Top 25 Most Dangerous Programming Errors.

These common weaknesses span three programmer error categories. Table 5 shows the distribution of attack outcomes for each error category. Table 5a further elaborates the specific common weaknesses for each error category.

Bad Actor Behavior

Steven Chabinsky, FBI deputy assistant director in the bureau's cyber division, understands the behavior of cyber criminals and reports that they have become highly accomplished specialists in their trade. These specialists are coordinated and can act fast performing a complex cyber criminal act from conception to completion in 24 hours. These specialties include:

- 1.** Coders or programmers who write malware and exploits
- 2.** Distributors or vendors who trade and sell stolen data
- 3.** Techies who maintain the needed information technology infrastructures
- 4.** Hackers
- 5.** Fraudsters who create social engineering schemes
- 6.** Hosters
- 7.** Money movers
- 8.** Launderers of digital proceeds
- 9.** People, often without technical skills, who handle personnel issues
- 10.** Leaders

Table 5a: 2010 Common Weakness/Attack Outcome

Common Weakness/Attack Outcome	Unauthorized Access	Loss of Data	Tampering with Data	Erosion of Performance	Denial of Service
Insecure Interaction Between Components	4	7	3	2	1
CWE-79 Failure to Preserve Web Page Structure ('Cross-site Scripting')	yes	yes		yes	
CWE-89 Improper Sanitization of Special Elements used in an SQL Command ('SQL Injection')		yes	yes		
CWE-352 Cross-Site Request Forgery (CSRF)	yes	yes			
CWE-434 Unrestricted Upload of File with Dangerous Type	yes	yes	yes		
CWE-78 Improper Sanitization of Special Elements used in an OS Command ('OS Command Injection')		yes	yes		
CWE-209 Information Exposure Through an Error Message		yes			
CWE-601 URL Redirection to Untrusted Site ('Open Redirect')	yes	yes			
CWE-362 Race Condition				yes	yes
Resource Management	2	8	9	9	3
CWE-120 Buffer Copy without Checking Size of Input ('Classic Buffer Overflow')		yes	yes	yes	yes
CWE-22 Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')		yes	yes	yes	
CWE-805 Buffer Access with Incorrect Length Value		yes	yes	yes	
CWE-754 Improper Check for Unusual or Exceptional Conditions		yes	yes	yes	
CWE-98 Improper Control of Filename for Include/Require Statement in PHP Program ('PHP File Inclusion')		yes	yes	yes	
CWE-129 Improper Validation of Array Index		yes	yes	yes	
CWE-190 Integer Overflow or Wraparound		yes	yes	yes	
CWE-131 Incorrect Calculation of Buffer Size		yes	yes	yes	
CWE-494 Download of Code Without Integrity Check	yes		yes		yes
CWE-770 Allocation of Resources Without Limits or Throttling				yes	yes
Porous Defenses	6	5	6	0	1
CWE-285: Improper Access Control (Authorization)	yes	yes	yes		
CWE-807 Reliance on Untrusted Inputs in a Security Decision	yes	yes	yes		
CWE-311 Missing Encryption of Sensitive Data	yes	yes	yes		
CWE-798 Use of Hard-coded Credentials	yes	yes	yes		
CWE-306 Missing Authentication for Critical Function	yes		yes		
CWE-732 Incorrect Permission Assignment for Critical Resource		yes			yes
CWE-327 Use of a Broken or Risky Cryptographic Algorithm	yes		yes		
Total yes	12	20	18	11	5

Detecting, apprehending, and successfully prosecuting these cyber criminals requires understanding the footprint of indicators associated with each and how to forensically connect their actions to a coordinated activity. Consequently the necessary capability within a police department to combat cyber crime is highly sophisticated and specialized and constantly evolving.

Conclusion

It is essential that the nation have a cyber strategy with a policy authorizing, in advance, time-critical actions based on well-balanced tradeoffs that have carefully weighed cause, effect, and consequences.

It is essential that each industry sector and enterprise appraise the cyber security goals to which it is committed, standard of excellence attributes which it can rely on, and the attack

outcomes and bad actors to which it is particularly vulnerable in preparation for ranking consequences to determine what can be traded off if push comes to shove.

It is essential that each industry sector and enterprise possess coordinated and deployable cyber tactics for anticipation, detection, attribution, and counter measures.

Disclaimer

Copyright © Don O'Neill 2010. Don O'Neill retains copyrights to this material.

Permission to reproduce this document and to prepare derivative works from this document for internal use is granted, provided the copyright and "No Warranty" statements are included with all reproductions and derivative works.

ABOUT THE AUTHOR



Don O'Neill is a seasoned software engineering manager and technologist. Following his 27 year career with IBM's Federal Systems Division, Mr. O'Neill completed a three-year residency at Carnegie Mellon University's SEI under IBM's Technical Academic Career Program and served as an SEI Visiting Scientist.

As an independent consultant, Mr. O'Neill conducts defined programs for managing strategic software spanning competitiveness, security, and process improvement. As an expert witness, he provides testimony on the state of the practice in developing and fielding large-scale industrial software and the complex factors that govern their outcome.

As an inventor, Mr. O'Neill has two patents pending. One, trademark registered Trusted Pipe™, is entitled "Business Management and Procedures Involving Intelligent Middleman," an apparatus and method for the inside track to offshore outsourcing. The other, trademark registered Smart Pipe™, is entitled "Business Management and Procedures Involving a Smart Pipe of Tiered Innovation Management Teams," an apparatus and method for harvesting ideas as intellectual property from knowledge workers on projects, whether onshore or offshore.

In his IBM career, Mr. O'Neill completed assignments in management, technical performance, and marketing in a broad range of applications including space systems, submarine systems, military command and control systems, communications systems, and management decision support systems. He was awarded IBM's Outstanding Contribution Award three times.

®Trusted Pipe is registered with the U.S. Patent and Trademark Office.

®Smart Pipe is registered with the U.S. Patent and Trademark Office.



U.S. Department of Defense *Systems Engineering*



INNOVATION, SPEED, AND AGILITY

U.S. Department of Defense applies best engineering practices to

- Support warfighter operations; manage risk with discipline
- Grow engineering capabilities to address emerging challenges
- Champion systems engineering as a tool to improve acquisition quality
- Develop future technical leaders across the acquisition enterprise

Dedicated to improving defense systems engineering to support the warfighter and ensure excellence in the defense acquisition workforce.

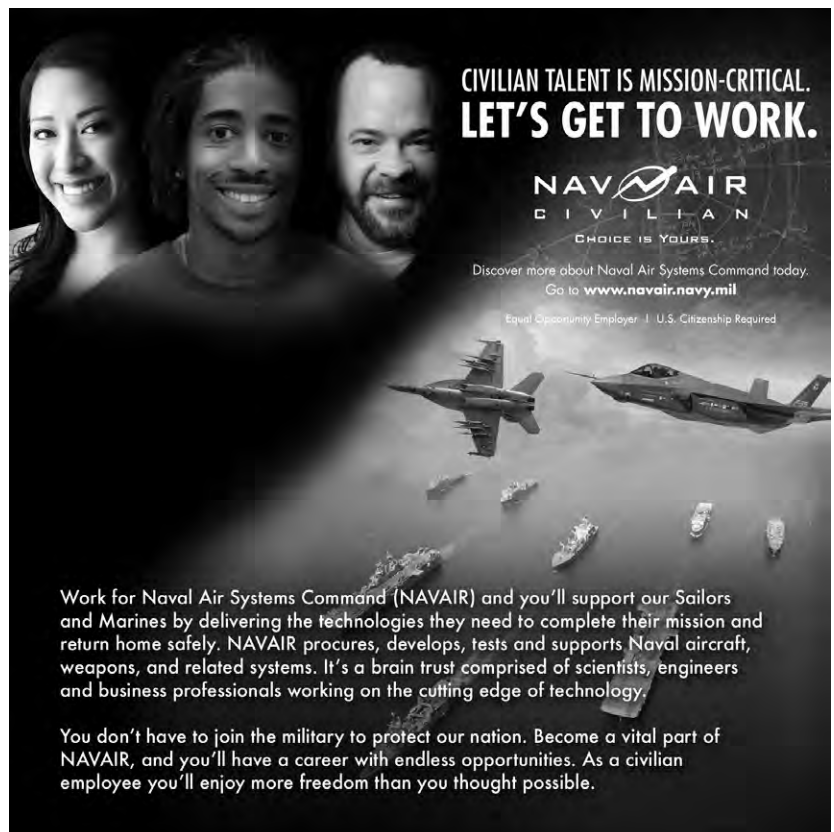
Deputy Assistant Secretary of Defense for Systems Engineering
3040 Defense Pentagon • Washington, DC 20301-3040 • <http://www.acq.osd.mil/se>

REFERENCES

1. Prieto, Daniel B. and Dr. Steven Bucci, "Meeting the Cyber Security Challenge: Empowering Stakeholders and Ensuring Coordination", IBM U.S. FederWhite Paper, 2010.
2. Davenport Thomas H., Sirkka L. Jarvenpaa, "Strategic Use of Analytics in Government", IBM Center for The Business of Government, 2008.
3. Blair, Dennis C., Director of Intelligence, "Annual Threat Assessment of the U.S. Intelligence Community for the Senate Select Committee in Intelligence", 2 February 2010.
4. O'Neill, Don, "Cyber Anticipation Tactics: Tracing Essential Factors of Security In-depth Tradecraft From Cause to Effect", Build Security In web site, Operated by Software Engineering Institute and Sponsored by Department of Homeland Security, to appear.
5. O'Neill, Don, "Peer Reviews", Encyclopedia of Software Engineering- Volume 2, Second Edition, Edited by John Marciniak, John Wiley & Sons, Inc., pp. 929-945.
6. Hevner, Alan R., Richard C. Linger, Rosann W. Collins, Mark G. Pleszkoch, Stacy J. Prowell, and Gwendolyn H. Walton, "The Impact of Function Extraction Technology on Next-Generation Software Engineering", Carnegie Mellon University CERT, CMU/SEI-2005-TR-015, July 2005.
7. Wheeler, David A. and Gregory N. Larsen, "Techniques for Cyber Attribution", Institute for Defense Analysis, IDA paper P-3792, October 2003.
8. Davenport Thomas H., Sirkka L. Jarvenpaa, "Strategic Use of Analytics in Government", IBM Center for The Business of Government, 2008.
9. "2010 CWE/SANS Top 25 Most Dangerous Programming Errors", Common Weakness Enumeration, The MITRE Corporation, February 17, 2010, <<http://cwe.mitre.org/top25>>.
10. O'Neill, Don and Albert L. Ingram, "Software Inspections Tutorial", Software Engineering Institute Technical Review 1988, pp. 92-120.
11. Schulmeyer, G. Gordon, "Handbook of Software Quality Assurance", Artech House, Inc. 2008, ISBN-13: 978-1-59693-186-2, 464 pages.

NOTES

1. Lais, Sami, "Cyber Crime Takes a Bite Out of Legit Systems", Washington Technology, The Magazine for Government Contractors, February 2010, page 24
2. Originated by Council on Competitiveness, Washington, D.C.
3. Palmisano, Samuel J., "A Letter From the Chairman", IBM 2009 Annual Report, page 6



**CIVILIAN TALENT IS MISSION-CRITICAL.
LET'S GET TO WORK.**

NAVAIR
CIVILIAN
CHOICE IS YOURS.

Discover more about Naval Air Systems Command today.
Go to www.navair.navy.mil

Equal Opportunity Employer | U.S. Citizenship Required

Work for Naval Air Systems Command (NAVAIR) and you'll support our Sailors and Marines by delivering the technologies they need to complete their mission and return home safely. NAVAIR procures, develops, tests and supports Naval aircraft, weapons, and related systems. It's a brain trust comprised of scientists, engineers and business professionals working on the cutting edge of technology.

You don't have to join the military to protect our nation. Become a vital part of NAVAIR, and you'll have a career with endless opportunities. As a civilian employee you'll enjoy more freedom than you thought possible.



Visit <http://www.crosstalkonline.org/events> for an up-to-date list of events.

Diminishing Manufacturing Sources & Material Shortages (DMSMS) & Standardization 2011

29 August – 1 September 2011

Ft. Lauderdale, FL

<http://www.acq.osd.mil/se/events>

45th Engineering & Technical Management (ETM) Conference

11-15 September 2011

St. Louis, MO

<http://www.acq.osd.mil/se/events>

AUTOTESTCON 2011

12-15 September 2011

Baltimore, Maryland

<http://www.autotestcon.com>

SwA Forum - Fall 2011

12-16 September 2011

Arlington, VA

<https://buildsecurityin.us-cert.gov/bsi/events.html>

TSP Symposium 2011

19-22 September 2011

Atlanta, GA

<http://www.sei.cmu.edu/tspsymposium/2011>

International Conference on Software and Knowledge Engineering

28-30 September 2011

Singapore

<http://www.waset.org/conferences/2011/singapore/icske>

14th Annual NDIA Systems Engineering Conference

24-27 October 2011

San Diego, CA

<http://www.acq.osd.mil/se/events>

MILCOM 2011

7-10 November 2011

Baltimore, MD

<http://www.milcom.org>

SwA Working Group Sessions - Winter 2011

28 November - 2 December 2011

McLean, VA

<https://buildsecurityin.us-cert.gov/bsi/events.html>

Software Assurance Forum - Spring 2012

26-30 March 2012

McLean, VA

<https://buildsecurityin.us-cert.gov/bsi/events.html>

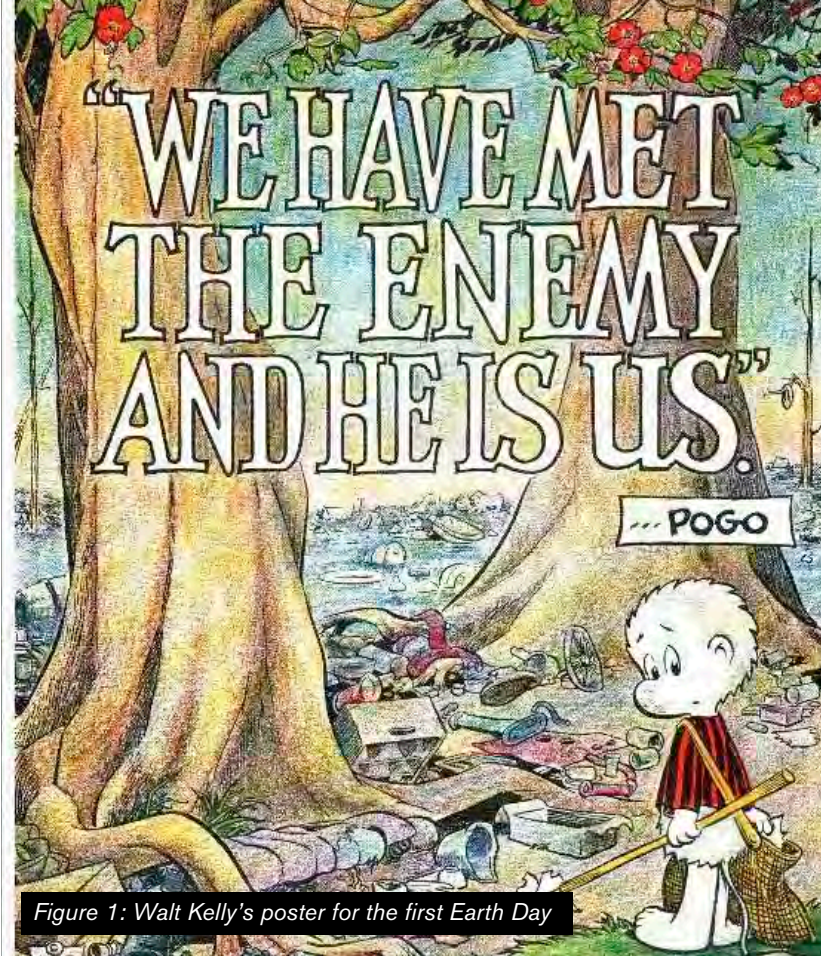


Figure 1: Walt Kelly's poster for the first Earth Day

Who Is Preying Upon Whom?

I am a frequent visitor to a local coffee vendor. While I prefer a relatively simple brew (please, give me a large cup of a caffeinated black liquid), my wife is a bit more elegant (she gets a triple venti, six pump, one Splenda, skinny hazelnut latte; or what I refer to as a diet candy bar in a cup). To save money, I have invested in a coffee card that gives me a 10% discount.

However, the other day I noticed an anomaly in the checkout system. If I use the card, I get a 10% discount. If I add money to the card, and then, on the same transaction buy coffee, it does not apply the discount.

I mentioned this to the folks behind the counter and they readily agreed that this seems to be a “known” problem. On days where an experienced counter person was working, they knew to do it as a two-step process—add money, then charge me for the coffee. However, new folks do not understand the system yet and do it all as a single transaction. The result, no discount.

In fact, the barista working last night pointed out to me that their new system had lots of “known errors.” In fact, in the training class for the new system, a full afternoon was reserved for “work-arounds.” The barista also pointed out that the old system, which used a non-graphical interface, worked almost perfectly. And, like generations of computer users before, upon getting the new system, they realized that the old system was much better.

Fred Brooks, in his classic (and “must read”) book, *The Mythical Man Month*, coined the term “Second-System Effect.” The second-system effect refers to the tendency of small, elegant, and successful systems to have elephantine, feature-laden monstrosities as their successors. Brooks initially used it to describe the jump from the elegantly simple operating system on the

IBM 700/7000 series to the extremely complex and incredibly difficult-to-master IBM OS/360. The second-system effect occurs because designers and developers try to put in the second system all the things they did not get to do first time, loading the second system up with all the features they put off while making version one. In reality, most of these “new features” should be put off in version two (and three, four, etc.) as well. I propose a generalization of this law (and let us modestly call it “Cook’s Law”)—the first version of any software system can be bad. However, only subsequent versions can be truly horrible. Perhaps even this could be generalized to, “The greater the version number, the greater the chance the software is bloated and has unneeded and unusable features.”

In a recent workshop at the Systems and Software Technology Conference, I got to speak on the skills necessary for a systems engineer. I used the term “The Zen of Systems Engineering.” This term refers to the extreme difficulty of building a simple, elegant system. Perhaps a corollary to Brooks’ Law should be that second systems tend to be bloated and feature-laden monstrosities, even when the developers are fully aware of the second-system effect. (Which, in turn, is just a restatement of Hofstadter’s Law: It always takes longer than you expect, even when you take into account Hofstadter’s Law.)

So, what can we do to make things better? Well, for one thing, we have to aggressively work with the end users to understand what features are needed and what features are merely desired. Any time a requirement includes the words, “It would be nice if...” quit writing. Wait for the words “We have to ...” Second systems will always be more complex (I have never seen a second system whose purpose was to remove features). But they should not be bloated and elephantine. Leonardo da Vinci said, “Simplicity is the ultimate sophistication.” Try to put in only what the users actually need. Keep delaying the “bells and whistles” as long as you can. Make the bloated and elephantine systems somebody else’s problem.

From the 1940s up until his death in the 1970s, the cartoonist Walt Kelly wrote a wonderful cartoon strip entitled *Pogo*. While funny in its own right, it frequently engaged in social and political satire, and therefore unless you understand the issues of the time, much of the humor is lost. In 1953, in a series of political satires attacking McCarthyism, he came up with the quote, “We have met the enemy, and he is us.” This quote was also used in a poster Walt Kelly created for the very first Earth Day in 1970.

“We have met the enemy and he is us.”

Wow. Kind of makes “Cook’s Law” pale by comparison, huh?

P.S. I was serious. If you develop software for a living, and you have not read *The Mythical Man Month*, go find a copy and read it now. *The Mythical Man-Month: Essays on Software Engineering* is widely regarded as a classic on the human elements of software engineering.

David A. Cook, Ph.D.
Stephen F. Austin State University
 cookda@sfasu.edu



Homeland
Security

Software Assurance

Software is essential to enabling the nation's critical infrastructure.

To ensure the integrity of that infrastructure, the software that controls and operates it must be secure and resilient.

Software Assurance Community Resources and Information Clearinghouse provides corroboratively developed resources. Learn more about relevant programs and how you can become involved.

Security must be "built-in" and supported throughout the lifecycle.

Visit <https://buildsecurityin.us-cert.gov> to learn about the practices for developing and delivering software to provide the requisite assurance. Sign up to become a free subscriber and receive notices of updates.

The Department of Homeland Security provides the public-private collaboration framework for shifting the paradigm to software assurance.



<https://buildsecurityin.us-cert.gov/swa>



NAV  AIR



CROSSTALK thanks the above organizations for providing their support.